Uso de redes sub-GHz para la gestión remota de *beacons*

TRABAJO DE FIN DE MÁSTER DEL MÁSTER EN INTERNET DE LAS COSAS

Adolfo Javier Machín Fernández

Departamento de Arquitectura de Computadores y Automática
Facultad de Informática
Universidad Complutense de Madrid

Convocatoria Junio 2018
Calificación: 10

Documento maquetado con TEXIS v.1.0+.

Este documento está preparado para ser imprimido a doble cara.

Uso de redes sub-GHz para la gestión remota de *beacons*

Curso académico 2017/2018
Máster en Internet de las Cosas

Dirigida por el Doctor
Francisco D. Igual Peña

Colaborador externo
Aurelio A. Cano Abellán, Everis

Departamento de Arquitectura de Computadores y Automática
Facultad de Informática
Universidad Complutense de Madrid

Convocatoria Junio 2018
Calificación: 10

Agradecimientos

Quiero agradecer en primer lugar a mi familia, ya que sin ella esto no habría podido ser posible. A mi pareja por estar ahí para ayudarme y aconsejarme cuando lo necesitaba, y aguantarme cuando era necesario. También quiero agradecer a mi tutor en *Everis* durante el desarrollo de este TFM, Aurelio A. Cano Abellán, por aconsejarme y dirigirme en la dirección correcta cuando lo he necesitado. Y por último, quiero agradecer en especial al Dr. Francisco D. Igual Peña por la confianza depositada en mi y la ayuda prestada en las correcciones de esta memoria.

Autorización

El abajo firmante, matriculado en el Máster en Internet de las Cosas de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Uso de redes sub-GHz para la gestión remota de *beacons*", realizado durante el curso académico 2017-2018 bajo la dirección de Francisco D. Igual Peña, y con la colaboración externa de dirección de Aurelio A. Cano Abellán, en el Departamento de Arquitectura de Computadores y Atomática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en internet y garantizar su preservación y acceso a largo plazo.

Adolfo Javier Machín Fernández.

Resumen

Hoy en día, los dispositivos móviles representan una parte muy importante de nuestras vidas cotidianas. Su importancia es tal que los usamos en todos los ámbitos de nuestra vida y ya no podemos imaginas una vida sin ellos. Gracias a esta evolución, se abren ante nosotros una gran cantidad de posibilidades sin explorar hasta ahora en diferentes ámbitos de nuestra vida.

Gracias a la aparición de los *beacons*, combinados con nuestros teléfonos móviles, se abren nuevas formas de interactuar con nuestro entorno: recibir nuevas ofertas cuando entramos a una tienda, obtener información acerca de la ocupación de un restaurante y su tiempo de espera y un sinfín de aplicaciones más.

Una de las principales problemáticas que presentan estos dispositivos es su *provisionamiento*, ya que deben ser configurados mediante tecnología *Bluetooth Low Energy (BLE)*, lo que nos obliga a realizar este proceso de provisionamiento y configuración de forma físicamente próxima al *beacon*.

En este TFM se hace uso de las ventajas de las redes sub-Ghz para facilitar la gestión y control remoto de *beacons* mediante el protocolo *Lightweight Machine To Machine (LWM2M)*. Para ello, se desarrolla una infraestructura basada en el servidor *Eclipse Leshan* para la gestión remota de despliegues de *beacons*, así como una adaptación del sistema operativo *Contiki* sobre placas *Launchpad* con soporte para doble banda (sub-Ghz y 2.4 Ghz).

Además, se exploran las capacidades de *Google Eddystone* y más en particular de *Eddystone URL* para controlar el tipo de información que emiten los *beacons*.

Palabras Clave

Beacons	Eddystone
LWM2M	sub-GHz
6LoWPAN	BLE
Contiki	Texas Instruments

Abstract

Nowadays, mobile phones have a great importance in almost all areas of our life. They play such an important role that we use them in almost every field of our day-to-day, and we cannnot imagine living without them. As a result, many possibilities in many fields of our daily life that remain unexplored are opened in front of us.

Due to the appearing of *beacons*, combined with our mobile phones, new ways of interaction with our environment are opened: receiving new sales when we come into a shop, getting information about the occupation of a restaurant and its wait time, among others.

One of the main problems about these devices is their provisioning and configuration process, as they must be provisioned with *Bluetooth Low Energy (BLE)* technology, which force us to be physically close to them in the process of provisioning and configuration.

This work makes use of the advantages of the sub-GHz networks to make it easier to manage and remote controlling beacons by using the *Lightweight Machine To Machine (LWM2M)* protocol. To achieve it, an infrastructure based on an Eclipse Leshan server will be developed to remotely manage beacons'deployments. An adaptation of the Contiki operating system over LaunchPad boards with dual band support is also performed.

In addition, the capabilities of *Google Eddystone*, and in particular *Eddystone URL*, to control the kind of information broadcasted by the beacon is explored.

Keywords

Beacons	Eddystone
LWM2M	sub-GHz
6LoWPAN	BLE
Contiki	Texas Instruments

Índice

Agradecimientos v

Autorización vii

Resumen ix

Abstract xi

1. Introducción 1

1.1. Motivación 1

1.2. Objetivos y organización del trabajo 2

1.3. Estructura de la memoria 4

2. Introduction 5

2.1. Motivation 5

2.2. Objectives and work organization 7

2.3. Structure of the document 8

3. Estado del arte 9

3.1. Redes inalámbricas 9

3.2. Systems On Chip 17

3.3. Sistemas Operativos RTOS 19

3.4. Protocolos para gestión de *beacons* BLE 21

3.5. Lightweight Machine To Machine (LWM2M) 26

4. Desarrollo del proyecto 29

4.1. Selección de tecnologías hardware, software y de red 29

4.2. Organización del proyecto 33
4.3. Modificaciones en Contiki 35
4.4. Router IP64 39
4.5. Implementación de beacons en Contiki 40
4.6. Resultado del desarrollo y caso de uso 43

5. Aplicaciones **47**
5.1. Marketing 47
5.2. Gestión de salas de reuniones 49

6. Conclusiones, aplicaciones y trabajo futuro **51**
6.1. Conclusiones 51
6.2. Trabajo futuro 52

7. Conclusions and future work **53**
7.1. Conclusions 53
7.2. Future work 54

Bibliografía **55**

Acrónimos **57**

Índice de figuras

1.1. Diagrama esquemático del sistema completo. 3

2.1. Complete system schematic diagram. 6

3.1. *Diagrama de bloques CC13XX.* 18
3.2. *Eddystone UID.* 23

4.1. *Launchad*, *SensorTag* y *DevPack.* 30
4.2. Topología de red desplegada. 34
4.3. Diagrama de flujo *LWM2M.* 36
4.4. *Router IP64* con el módulo *ENC28J60-H* y la placa *TI CC1310 LaunchPad.* 39
4.5. Notificación *Nearby* con la *URL* de un *beacon* cercano. 44
4.6. Lista de nodos y vista de un nodo controlando un *beacon* en *Leshan.* 45

5.1. Despliegue de *beacons* en un centro comercial. 48
5.2. Despliegue de *beacons* móviles en una ciudad. 49

Índice de Tablas

3.1. Ejemplo tabla *GATT*. 15
3.2. Formato de trama *Eddystone*. 22

4.1. Comparación pila IP y pila *6LoWPAN*. 32
4.2. Formato trama *Eddystone URL*. 42
4.3. Codificación de prefijos y sufijos en *Eddystone URL*. 43

Capítulo 1

Introducción

1.1. Motivación

Hoy en día, los dispositivos móviles representan una parte muy importante de nuestras vidas cotidianas. Atrás quedaron los tiempos donde estos dispositivos únicamente servían para realizar llamadas. A día de hoy, los usamos como reproductor de música, navegador web, agenda, calculadora, etc. Tal es su importancia en nuestro día a día, que ya no podemos imaginar una vida sin ellos. Debido a esta evolución, se abren ante nosotros una gran cantidad de posibilidades sin explorar hasta ahora en diferentes ámbitos de nuestra vida.

Gracias a la aparición de los *beacons* (Google, 2018b), pequeños dispositivos de bajo consumo que emiten señales *broadcast* mediante la tecnología *Bluetooth Low Energy (BLE)* (Townsend, 2015b), combinados con nuestros teléfonos móviles, se abren nuevas formas de interactuar con nuestro entorno: recibir nuevas ofertas cuando entramos a una tienda, obtener información acerca de la ocupación de un restaurante y su tiempo de espera, nuevas formas de interactuar con obras de arte en museos pudiendo obtener más información acerca de lo que estamos viendo y un sinfín de aplicaciones más.

Una de las principales problemáticas que presentan estos dispositivos es su *provisionamiento*, ya que deben ser configurados mediante tecnología *BLE*, lo que nos obliga a realizar este proceso de provisionamiento y configuración de forma físicamente próxima al *beacon*, por ejemplo para (re)configurar la información que deseamos difundir.

El presente trabajo propone una solución a esta problemática introduciendo una infraestructura que permita gestionar los *beacons* de manera remota y centralizada, pudiendo provisionarlos de los contenidos que tienen que emitir sin necesidad de estar físicamente cerca de ellos. En el sistema propuesto, mediante el uso de dispositivos con doble banda, los *beacons* son

capaces de emitir tramas *broadcast* mediante *BLE*; además, con su banda de radio sub-GHz es posible extender el rango de alcance que tienen los dispositivos.

Asimismo, otro problema que puede surgir al crear una red con dispositivos que se encuentran en entornos no controlados, como puede ser un centro comercial, y que además funcionan con batería y que por lo tanto puede consumirse, es que los nodos pueden desaparecer de la red y por tanto dejar de prestar conectividad al resto de nodos que conforman la red. Este problema es subsanable mediante el uso de redes malladas (Wikipedia, Mesh networking), con las que garantizamos que la comunicación entre el servidor, encargado de realizar el provisionamiento, y el *beacon* no se pierda aunque algunos de los nodos de la red no estén disponibles en todo momento.

Por otra parte, ya que el sistema que se propone debe ser flexible para poder adaptarse a las configuraciones que red que se encuentran ya desplegadas en los entornos de implantación de los *beacons*, es necesario que exista una conectividad punto a punto sin importar el tipo de red que haya fuera de la red mallada. Mediante el protocolo *6LoWPAN* (Wikipedia, 6LoWPAN), se consigue una conectividad punto a punto entre un nodo de la red mallada y un dispositivo del exterior sin importar el tipo de red del que se trate.

Así, en la Figura 1.1 se puede observar un esquema general de la infraestructura propuesta, formada por *beacons* dispuestos en una red *6LoWPAN*, interconectados a través de su interfaz sub-GHz. Como *gateway* dispondremos de una placa con conectividad sub-GHz y un módulo *Ethernet* acoplado que se encargará de realizar el intercambio de paquetes entre las redes y la conversión de *IPv4* a *IPv6*. Los *beacons* se podrán controlar remotamente desde un servidor *Lightweight Machine To Machine (LWM2M)* (Alliance, 2014) y emitirán información siguiendo el protocolo de *Eddystone URL* (Google, 2016), como se detallará más adelante.

1.2. Objetivos y organización del trabajo

El objetivo principal del proyecto radica en proponer una solución integral para resolver las problemáticas anteriormente mencionadas relacionadas con el uso de despliegues de *beacons*: su provisionamiento y todos los desafíos añadidos que conlleva, como la creación de una red resistente a caídas y la conectividad punto a punto en dicha red sin importar la topología de red desplegada fuera de ésta. Para esto se ha hecho uso de las tecnologías también mencionadas anteriormente. De forma colateral y como necesidad del proyecto, se añadirá mayor funcionalidad al sistema operativo *Contiki* (Thingsquare, 2006) (véase 3.3.2), ya que actualmente no implementa ningún protocolo de emisión de datos para *beacons*.

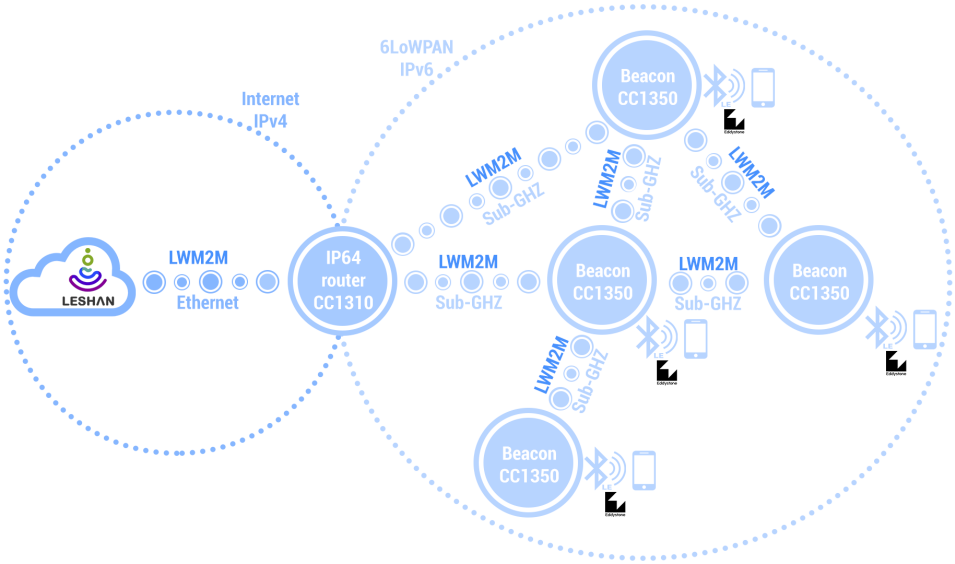


Figura 1.1: Diagrama esquemático del sistema completo.

Para ello, se dividió el trabajo en los siguientes pasos básicos:

Estudio y selección de hardware. En primer lugar se realizó un estudio en profundidad del *hardware* disponible en el mercado, eligiendo aquel que se adaptase mejor a las necesidades del proyecto.

Selección de sistema operativo para *beacons* y *gateway*. Posteriormente, con el *hardware* ya elegido se decidió el sistema operativo que se iba a utilizar a nivel de *router* y *beacons*.

Despliegue y configuración de la red mallada. Una vez elegido el *hardware* y el sistema operativo, se diseñó una topología de red mallada con los dispositivos haciendo uso del protocolo de enrutamiento *Routing over Low Power and Lossy Networks (RPL)* (Alliance, 2011) y con *6LoWPAN* en la capa de adaptación.

Configuración del *gateway*. Para que la red tuviera conectividad punto a punto era necesario un *gateway* que conectara la red mallada con el exterior. Para ello se desarrolló un *router* basado en las placas que estábamos utilizando y acoplándole un módulo *Ethernet*. Éste es el encargado de realizar la traducción de paquetes entre las distintas redes.

Estudio e implementación del protocolo de emisión de datos en Contiki. Posteriormente, se escogió el protocolo para la emisión de datos para los *beacons*. El protocolo elegido fue *Eddystone* (Google, 2018a) y se

implementó su versión *Eddystone URL* en el sistema operativo elegido, *Contiki*.

Desarrollo de la infraestructura de gestión centralizada. Por último, se decidió usar *LWM2M* para gestionar los *beacons* de forma centralizada desde un servidor. Se implementó una versión de *LWM2M* que se adaptaba a nuestras necesidades y los *IPSO Smart Objects* (Alliance, 2018) necesarios para la gestión de los *beacons*.

En la presente memoria se expondrán tanto las razones tras la elección de cada elemento de la infraestructura como las alternativas adicionales valoradas durante el proceso de estudio inicial.

1.3. Estructura de la memoria

El documento está distribuido en seis capítulos claramente diferenciados entre ellos, en los que se detalla tanto el planteamiento del proceso seguido en la elaboración del proyecto como el desarrollo del mismo.

En el primer capítulo se describe la motivación que se siguió a la hora de realizar este proyecto, así como los objetivos que se querían alcanzar. El segundo capítulo es una traducción al ingles del primero.

En el tercer capítulo se propone un análisis completo de las tecnologías relacionadas con el proyecto y que se barajaron a la hora de decidir cuales se usarían para su desarrollo.

El cuarto capítulo se centra en desarrollo del proyecto en sí y en la implementación de las distintas partes que lo componen dentro del sistema operativo elegido.

En el quinto capítulo se describen algunas posibles aplicaciones del proyecto en el mundo real.

En el sexto capítulo se exponen las principales conclusiones del trabajo realizado y se detallan las posibles mejoras que se pueden incorporar en futuras versiones del proyecto. El séptimo capítulo es una traducción al ingles del sexto.

Capítulo 2

Introduction

2.1. Motivation

Nowadays, mobile phones play an important role in almost every area of our life. The days when these devices were only used to make phone calls have gone. As of today, we employ them as a music player, web browser, calendar, calculator, etc. They are such an important part of our dayly life that we cannot imagine living without them anymore. Because of this revolution, many possibilities in many fields of our daily life worth exploring are opened in front of us.

Due to the appearance of *beacons*, small low power devices that broadcast frames though *Bluetooth Low Energy (BLE)*, combined with our mobile phones, new ways for interacting with our environment are opening: receiving new sales when we go into a shop, get information about the occupation of a restaurant and its wait time, or new ways to interact with artworks in museums being able to get more information about the pieces we are seeing, among many others.

One of the main problems regarding these devices is their provisioning mechanisms, as they must be provisioned with *BLE* technology, which force us to be physically close to them during the process, dramatically limiting their flexibility and ease of deployment.

This project proposes a solution to overcome this limitation by introducing an infrastructure that allows a centralized and remote control of a set of beacons without the need of being physically close to them. The proposed system leverages dual band devices, where beacons are able to send broadcast frames through *BLE*, and additionally with its sub-GHz radio band it is possible to extend its range.

Likewise, the problem that may emerge when creating the network with

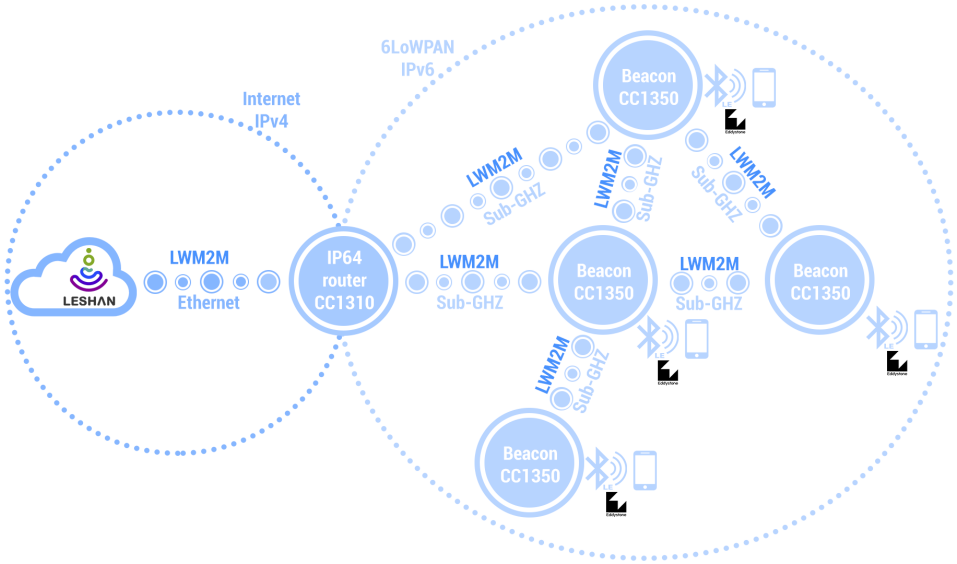


Figura 2.1: Complete system schematic diagram.

devices that are in a non-controlled environment, e.g. a shopping mall, and that works with batteries is that nodes may disappear from the network, making it impossible to to reach the rest of the network communicated through them. This problem can be solved by using *mesh networks*. With them, we guarantee communication between server and nodes even if some of them are not available at all times.

On the other hand, since the system is going to be deployed in environments with a certain network infrastructure, it must be flexible enough to adapt to them. In addition, it must exist a point-to-point connectivity no matter the network topology deployed outside the mesh network. With the *6LoWPAN* protocol, we can achieve this point-to-point connectivity between a node inside the mesh network and a device outside it independently from the kind of network used outside.

Thus, we can see a general scheme of the infrastructure in figure 2.1. Beacons are making up a *6LoWPAN* mesh network with their sub-GHz interface. As a gateway, we have a board with sub-GHz connectivity and an Ethernet module attached to it. This module is in charge of the package exchanges between networks and the conversion from *IPv4* to *IPv6*. Beacons can be controlled remotely from a *LWM2M* server and they will broadcast the information with *Eddystone URL* protocol that will be explained later.

2.2. Objectives and work organization

The main objective of the project is to propose a solution to solve the limitations previously mentioned and related to the beacon deployment process. Collaterally and as a need of the project, functionality has been added to Contiki (see 3.3.2) as it currently does not implement any beacon broadcasting protocol.

For it, the work was divided in the following steps:

Research and hardware selection. First, a deep research effort about available hardware in the market was carried out, and the best suited the project needs was selected.

Operating system for beacons and gateway selection. Later, with the hardware already selected, the operating system for the beacons and the gateway was chosen.

Mesh network deployment and configuration. Once hardware and operating system were selected, the network topology was designed for the devices using the routing protocol *RPL* and *6LoWPAN* in the adaptation layer.

Gateway configuration. To achieve point-to-point connectivity in the network, a gateway to connect it with the exterior was needed. For it, a router was developed based on the boards that were being used and an Ethernet module was attached to it. This gateway is in charge of translating the packages between networks.

Data transmission protocol research and implementation in Contiki. Later, the beacon broadcast protocol was selected. The selected procotol was *Eddystone*, and its version *Eddystone URL* was implemented in *Contiki*.

Centralized management infrastructure development. Finally, *LWM2M* was chosen to manage the beacons centrally from the server. A *LWM2M* version was implemented to fit our needs and the IPSO Smart Objects required for the beacons manage were implemented too.

In this document, the reasons for the choice of each element of the infrastructure and the additional alternatives evaluated during the initial study process will be explained.

2.3. Structure of the document

This document is organized in organized in seven chapters clearly differentiated where it is shown the proposal and process of the project and the development of the project itself.

The first chapter motivates the work and describes in details the objectives we want to achieve. The second chapter is the translation into English of the first chapted.

In the third chapter, a deep analysis of related technologies considered at the time is done and the reasons for each decision are explained.

The fourth chapter is focused on the project development itself and the implementation of all the components that conform the chosen operating system.

In the fifth chapter, we describe some real world applications of the project of the developed infrastructure.

The sixth chapter presents the main conclusions of the word done and is shows the possible improvements that can be introduced in future version of the project. The seventh chapter is the translation into English of the sixth chapter.

Capítulo 3

Estado del arte

En este capítulo se realiza una descripción detallada de los protocolos y sistemas relevantes en el ámbito del proyecto, haciendo especial hincapié en aquellos aspectos o características de los mismos que se consideran de interés para la correcta comprensión del trabajo realizado. Esta introducción pretende servir al lector para poner en situación de los protocolos y sistemas existentes actualmente, así como para servir de comparativa entre las ventajas y desventajas que estos presentan.

3.1. Redes inalámbricas

Las redes inalámbricas o *wireless* son un tipo de redes en el que la conexión entre nodos se realiza por medio de ondas electromagnéticas, sin necesidad de una red cableada.

Según su cobertura se pueden clasificar en diferentes tipos:

- *Wireless Personal Area Network (WPAN)*: conectan dispositivos en áreas relativamente pequeñas. Aquí se encontrarían tecnologías como *Bluetooth*.
- *Wireless Local Area Network (WLAN)*: proveen interconexión entre dispositivos en distancias cortas, normalmente proveyendo de conexión a Internet través de un punto de acceso compartido. Engloban productos bajo el estándar *IEEE 802.11* bajo la marca de *Wi-Fi*.
- *Wireless Metropolitan Area Network (WMAN)*: conectan diferentes redes *WLAN*. Un ejemplo sería *WiMAX*.
- *Wireless Wide Area Network (WWAN)*: redes inalámbricas que cubren áreas grandes como las que hay entre barrios o ciudades.

3.1.1. Redes malladas

Las redes malladas (*mesh*) son un tipo de red implementadas sobre una red inalámbrica *LAN*. En este tipo de redes los nodos se conectan de forma directa, no dinámica y no jerárquica a tantos otros nodos como sea posible y cooperan entre sí para encaminar de manera eficiente los datos desde y hacia los clientes. Las redes malladas se autoorganizan y autoconfiguran dinámicamente.

Las redes malladas permiten que los dispositivos se comuniquen entre sí sin contar con un punto de acceso centralizado. Los dispositivos que las forman pueden no enviar directamente sus paquetes al punto de acceso sino que pueden intercambiarlos con otros dispositivos de red para que lleguen a su destino.

La principal ventaja que aporta esta característica es que garantizamos que la comunicación entre nodos no se pierda aunque algunos de los nodos de la red no estén disponibles en todo momento.

3.1.2. Tecnologías sub-GHz

Las redes Sub-GHz son aquellas redes inalámbricas que trabajan por debajo de la banda del GHz (típicamente, 868-928 MHz, 433 MHz y 315 MHz). Operando a frecuencias más bajas, se consiguen mayores rangos de alcance que con frecuencias mayores. Además, debido a esta baja frecuencia consume menos energía que las bandas con mayor frecuencia, ya que se necesita una menor energía en el emisor para obtener la misma señal de potencia de salida que en frecuencias mayores. Por último, cabe destacar que la banda de sub-GHz tiene menores interferencias ya que al transmitir en una banda de frecuencias tan baja hay menos aplicaciones que usen el mismo espectro. Además, las bandas *ISM* de frecuencia más baja traspasan mejor entre los edificios en entornos urbanos. Obviamente, estas características positivas suelen presentarse a cambio de un menor ancho de banda de transmisión, por lo que este tipo de redes suelen darse en entornos en los que se requieren emisiones de largo alcance a bajas velocidades de transmisión.

3.1.2.1. 6LoWPAN

IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) es un estándar que posibilita el uso de IPv6 sobre redes basadas en el estándar *IEEE 805.15.4*. Hace posible que los nodos de una red inalámbrica se conecten directamente con otros dispositivos mediante *IP*. Este estándar actúa como una capa de adaptación entre la tecnología estándar *IPv6* y el medio ofrecido por el *IEEE 802.15.4* para comunicaciones con pérdidas y de bajo

consumo. *6LoWPAN* deriva de *LoWPAN*, que es una instancia particular de *LNNs* (*Low-Power and Lossy Network*) formada con dispositivos compatibles con el IEEE 802.15.4. Las características de *LoWPAN* son:

- Adecuado para sistemas que tienen capacidad de proceso y memoria limitadas.
- Bajo consumo energético.
- Corto alcance.
- Bajo coste.

Estas características fijan unas restricciones a la hora de desarrollar bajo los estándares del *IEEE 802.15.4*, como por ejemplo:

- Tamaño de paquete reducido. El tamaño máximo de un *frame* es de 127 *Bytes*.
- Bajo ancho de banda. Entre 250 *kbps* y 20 *kbps*.
- Debe ser tolerante a fallos debido a las características de los dispositivos; por ejemplo, se encuentran en entornos donde la conectividad es inestable y además se suele tratar de dispositivos que funcionan con baterías que pueden eventualmente agotarse.
- Debe tener soporte para modos de reducción de consumo.

6LoWPAN permite el uso de *IPv6* en redes que cumples con las características anteriores. Sus objetivos y características son:

- Se encarga del proceso de fragmentación y reensamblado de paquetes. Debido a que *IPv6* limita el tamaño mínimo de paquete a 1280 *Bytes* pero *IEEE 802.15.4* puede producir únicamente paquetes de hasta 81 *Bytes*.
- Comprime los encabezados. Debido a que las cabeceras *IPv6* tienen un tamaño de 40 *Bytes* y, como se ha dicho antes, el tamaño máximo de *frame* en el *IEEE 802.15.4* es de 81 *Bytes* es necesario comprimir la primera para dejar espacio al resto del *frame*.
- *6LoWPAN* también se encarga de la autoconfiguración de direcciones lógicas.
- También se encarga de proporcionar protocolos de enrutamiendo dinámicos para redes *mesh*.

3.1.2.2. Routing Protocol for LLN (RPL)

Routing over Low Power and Lossy Networks (RPL) es un protocolo de enrutamiento diseñado específicamente para redes de bajo consumo y con pérdidas. Se trata de un protocolo de vector de distancias proactivo. En este tipo de protocolos, cada nodo conoce a los vecinos que se encuentran conectados con él y los costes de dichos enlaces. Cada cierto tiempo, el nodo transmite su tabla de enrutamiento a sus vecinos y éstos recalculan sus tablas si existe alguna nueva información. En estos protocolos los nodos no conocen la información de toda la topología de red, únicamente la de sus nodos vecinos. *RPL* organiza la topología como un conjunto de estructuras llamadas *Destination-Oriented Directed Acyclic Graphs (DODAGs)* con una única raiz denominada *DAG ROOT*. Las *DODAGs* se optimizan utilizando una función objetivo según una métrica elegida, que puede ser latencia, distancia, consumo de energía, etc. Cada nodo se identifica con un número de rango, valor que puede utilizarse para determinar su posición relativa y distancia respecto al nodo raiz.

En una misma red, pueden existir múltiples instancias independientes de *RPL*, pudiéndo un nodo pertenecer a diversas instancias de *RPL* y actuar como *router* en alguna de ellas y como nodo en otras. Por otro lado, un conjunto de *DODAGs* pueden pertenecer a una instancia *RPL*, y un nodo puede ser miembro de múltiples instancias *RPL*, pero únicamente puede pertenecer a, como máximo, un *DODAG* por instancia.

El tiempo entre intercambios de mensajes entre nodos para la construcción y actualización de las rutas viene determinado por un temporizador. Los mensajes que se intercambian para este cometido tienen el nombre de *DODAG Information Objects (DIOs)*. Además, un nodo puede solicitar información a sus vecinos enviado mensajes *DODAG Information Solicitations (DISs)*. Con estos mensajes solicita a los nodos vecinos información para poder actualizar su tabla de enrutamiento o bien para unirse a una nueva instancia.

Por último, cabe destacar que *RPL* dispone de dos mecanismos para reparar la topología de un *DODAG*. El primero de ellos sirve para evitar blucles y también para posibilitar que nodos se unan al *DODAG*. El segundo de ellos se denomina reparación global y únicamente puede ser iniciado por el nodo *root* del *DODAG*. Consiste en incrementar su número de versión y realizar una nueva versión renovada del *DODAG*.

3.1.3. Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) es un subconjunto restringido del protocolo *Bluetooth* introducido como parte de la especificación básica de *Bluetooth*

4.0. *BLE* opera en el mismo rango de frecuencias que el *Bluetooth* clásico (2,4 - 4,44835 GHz) pero utilizando otro conjunto de canales. En lugar de 79 canales de 1 MHz utiliza únicamente 40 canales de 2 MHz. Dentro de un canal, los datos se transmiten utilizando modulación por desplazamiento de frecuencia gausiana (Wikipedia, Modulación por desplazamiento de frecuencia gausiana), con un *bitrate* de 1 *Mbit/s* y una potencia máxima de 10 *mW*. Comparado con *Bluetooth* clásico, pretende proporcionar un consumo de energía y coste menor manteniendo el mismo alcance.

Bluetooth Low Energy está soportado de forma nativa por los siguiente sistema operativos: *iOS*, *Android*, *Windows Phone*, *BlackBerry*, *macOS*, *Linux Windows 8* y *Windows 10*.

3.1.3.1. GAP

Generic Access Profile (GAP) (Townsend, 2015a) es el encargado de controlar las conexiones y los anuncios en *BLE*. Es el mecanismo que hace visible un dispositivo compatible con *BLE* al mundo exterior, y determina el modo en el que dos dispositivos interactúan. *GAP* define varios roles para los dispositivos, que se dividen en dispositivos *centrales* y dispositivos *periféricos*. Los dispositivos centrales son los dispositivos con mayor potencia de cómputo y a los que se conectan uno o más dispositivos periféricos. En el otro lado se encuentran los dispositivos periféricos, que son dispositivos de pequeñas dimensiones, con recursos limitados, bajo consumo energético y que se conectan a un dispositivo central con mayores capacidades.

Existen dos mecanismos principales para enviar anuncios utilizando *GAP*: los datos de anuncio *Advertising Data* y los datos de respuesta a escaneo *Scan Response*. Ambos son idénticos en forma y tienen una longitud de hasta 31 *Bytes*. Los *Advertising Data* son obligatorios y son la información que se transmite constantemente desde el dispositivo periférico para informar a los dispositivos centrales de su existencia. Los *Scan Response* son datos secundarios en importancia y pueden ser demandados por los dispositivos centrales. Permiten añadir cierta información adicional al proceso de anuncio, como por ejemplo un nombre de dispositivo.

La forma en que funciona el proceso de anuncio en *BLE* es el siguiente: un dispositivo periférico establece un tiempo de anuncio concreto y cuando expira retransmite el paquete básico de anuncio. Además, si un dispositivo central está interesado en recibir una respuesta a escaneo puede solicitarla y el dispositivo periférico responderá con datos adicionales.

3.1.3.2. GATT

Generic Attribute Profile (GATT) (SIG, 2018) es el mecanismo utilizado por dos dispositivos *BLE* mediante el cual se transfieren datos en ambos sentidos. Hace uso de un protocolo llamado *Attribute Protocol (ATT)*, utilizado para almacenar servicios, características y datos relacionados en una tabla de búsqueda que utiliza un identificador (*ID*) para cada entrada de la tabla. *GATT* entra en funcionamiento una vez queda establecida una conexión (o emparejamiento) entre dos dispositivos, y por lo tanto, cuando ya ha finalizado el proceso gobernado por *GAP*.

Es necesario destacar que el concepto de conexión entre dispositivos es exclusivo, es decir, un dispositivo periférico únicamente puede estar conectado a un dispositivo central al mismo tiempo, aunque el dispositivo central sí puede estar conectado a varios dispositivos periféricos a la vez. Por esta razón, cuando un dispositivo periférico se conecta con un dispositivo central, el primero finaliza su periodo de anuncio, y otros dispositivos centrales serán incapaces de seguir observando sus anuncios o conectar con él hasta que la conexión establecida se rompa.

Los dispositivos periféricos son conocidos como servidores *GATT*, almacenan las tablas *ATT* y definen los servicios y características. Los dispositivos centrales son conocidos como clientes *GATT* y son los encargados de enviar peticiones al servidor. Todas las transacciones son iniciadas por el cliente *GATT*, que es el dispositivo maestro, que recibe respuestas desde el servidor, que es el dispositivo esclavo.

Al establecer una conexión entre ambos dispositivos, el servidor sugiere un intervalo de conexión al cliente. Éste intentará reconectar con el primero cada vez que expire este intervalo en busca de nuevos datos. Como se ha dicho, este tiempo es una sugerencia y no obliga al cliente a ceñirse a él en su proceso de consulta periódica.

Las transacciones *GATT* se basan en objetos de alto nivel con una estructura jerárquica llamados *perfiles*, *servicios* y *características*:

- Los *perfiles* son colecciones de servicios predefinidas por parte de *Bluetooth SIG*. Éstos realmente no existen en los dispositivos periféricos, sino que se trata de una abstracción para poder organizar los servicios.
- Los *servicios* son colecciones de características y relaciones con otros servicios, y que encapsulan el comportamiento de un dispositivo o parte del mismo. Dividen los datos en entidades lógicas llamadas características, y pueden poseer una o más. Cada servicio se identifica por medio de un identificador único llamado *UUID*.
- Las *características* representan el concepto de menor nivel en las tran-

Perfil de proximidad	
Servicio sensor volumétrico	
UUID	Lectura
Activar/Desactivar sensor	Lectura/Escritura
Valor del sensor	Lectura
Máximo valor sensor leido	Lectura
Servicio sensor infrarojo	
UUID	Lectura
Activar/Desactivar sensor	Lectura/Escritura
Valor del sensor	Lectura
Mínima precisión del sensor	Lectura
Perfil ritmo cardíaco	
Servicio sensor pulso de luz	
UUID	Lectura
Activar/Desactivar sensor	Lectura/Escritura
Valor del sensor	Lectura
Mínimo valor sensor leido	Lectura
Servicio sensor oxigenación en sangre	
UUID	Lectura
Activar/Desactivar sensor	Lectura/Escritura
Valor del sensor	Lectura
Hora última lectura	Lectura

Tabla 3.1: Ejemplo tabla *GATT*.

sacciones *GATT*. Encapsulan un único dato y se distinguen por medio de un *UUID*. Pueden ser usadas tanto para enviar datos a los dispositivos centrales por medio de una lectura, o para recibir datos de éstos por medio de una escritura.

Un ejemplo para entender mejor las tablas *GATT* sería el de la tabla 3.1, donde se describen dos perfiles con sus servicios y características.

3.1.3.3. *Beacons* BLE

Los *beacons* son dispositivos de bajo consumo y, normalmente, pequeñas dimensiones, que emiten señales *broadcast* por medio de *BLE* sin necesidad de sincronización con los receptores de este señal. Suelen estar formados por

un microcontrolador y un chip de radio *BLE*, generalmente fabricado a día de hoy por dos grandes empresas: *Texas Instruments* o *Nordic Semiconductor*.

Los *beacons* transmiten la señal con una potencia fija conocida como *Tx Power*. Según viaja la señal la potencia disminuye, por lo que con un *Tx Power* superior puede viajar más distancia aunque también implica un mayor consumo energético. La frecuencia de emisión de anuncios se conoce como *advertising interval* y fija la frecuencia con la que un *beacon* emite señales *broadcast*. De nuevo, a menor *advertising interval* se emiten más señales *broadcast* por lo que la duración de la batería disminuye.

Los *beacons* pueden ser provisionados mediante *GATT* usando *BLE*, lo que es su mayor limitación debido a que se debe estar cerca de ellos físicamente para que se establezca una conexión *BLE*, lo que puede suponer un problema a la hora de desplegar grandes despliegues de *beacons*.

Existen multitud de aplicaciones reales para los *beacons*, entre ellas, destaca la posibilidad de utilizarlos para la geolocalización de dispositivos en interiores. Esto se realiza colocando varios *beacons* que emiten señales *broadcast*. Cuando el dispositivo móvil del usuario recibe esta señal, envía una respuesta con la que los *beacons* calculan su *Received Signal Strength Indicator (RSSI)*. El *RSSI* es una escala para medir el nivel de potencia de las señales recibidas por un dispositivo en las redes inalámbricas. Este *RSSI* es enviado a un servidor que mediante trilateración calcula la posición del usuario respecto a los *beacons*. De esta manera, el usuario puede saber dónde se encuentra dentro de un entorno cerrado sin la necesidad de utilizar señal *GPS*, únicamente haciendo uso de *Bluetooth*.

Otro ámbito donde los *beacons* tienen una gran utilidad es en el *marketing*. Debido a sus pequeñas dimensiones, bajo consumo y a que funcionan sin necesidad de sincronización con los receptores, son idóneos para crear nuevas experiencias de usuario en comercios. Mediante su uso se pueden crear experiencias personalizadas anunciando nuevos productos de la marca, las últimas ofertas de un establecimiento o simplemente enviando un saludo a los clientes que entran por la puerta. Esto es tan sencillo como colocar *beacons* en lugares estratégicos desde donde emiten mensajes personalizados según las campañas publicitarias en vigor, y que llegan a los teléfonos móviles de los usuarios de manera inmediata.

Por último, cabe destacar también el uso de los *beacons* en la asistencia médica. Usando las posibilidades de geolocalización mencionadas anteriormente es posible controlar a los pacientes en el hogar, controlando sus movimientos y actividades. Son una buena alternativa a las cámaras tradicionalmente usadas para esto, ya que aumentan el nivel de privacidad del paciente.

3.2. Systems On Chip

Un *System On Chip (SoC)* (Wikipedia, System on a chip), es un circuito integrado que integra en el mismo encapsulado todos los componentes de un ordenador u otro sistema electrónico. Normalmente, suelen incluir una *CPU*, memoria y dispositivos para entrada/salida. En contraste con las arquitecturas tradicionales basadas en placas base, donde se separan los componentes y posteriormente se unen a través de ésta, los *SoC* integran todos estos componentes en un único circuito integrado, lo que se suele traducir en mejoras de rendimiento, consumo de energía, menor tamaño y costes inferiores. El diseño de éstos se realiza mediante la combinación de módulos (*CPUs*, *GPUs*, interfaces...), que se consideran *cajas negras*, comunicadas mediante buses. Aunque actualmente existen multitud de fabricantes en el mercado dedicados a los *SoC*, nos centraremos en *Texas Instruments* ya que serán las placas que utilizaremos para el desarrollo del proyecto.

3.2.1. Texas Instruments

Texas Instruments (TI) es una compañía tecnológica que diseña y fabrica semiconductores y circuitos integrados. Es una de las diez compañías más importantes a nivel global por volumen de trabajo. Su trabajo se centra en el desarrollo de sistemas de chips analógicos y procesadores embebidos.

La familia CC13xx de *TI* es una serie de dispositivos radio frecuencia de baja potencia y bajo consumo de *Texas Instruments* que se centran en el bajo consumo energético y en la transmisión de datos mediante tecnologías sub-GHz, lo que permite lograr una excelente autonomía de la batería y un gran rango de cobertura. Actualmente hay dos dispositivos que componen esta familia, el CC1310 y el CC1350.

3.2.1.1. CC1310

El CC1310 Instruments (2016a) es miembro de la familia de dispositivos CC13XX. Posee un transceptor de baja frecuencia, un procesador de 48 MHz Cortex-M3 y un controlador de radio dedicado (Cortex-M0). Además, incorpora un modo de ultra ahorro de energía capaz de manejar sensores tanto analógicos como digitales permitiendo a la *MCU* maximizar sus tiempos de reposo y así ahorrar energía.

Se puede ver un diagrama de bloques completo del chip en la figura 3.1

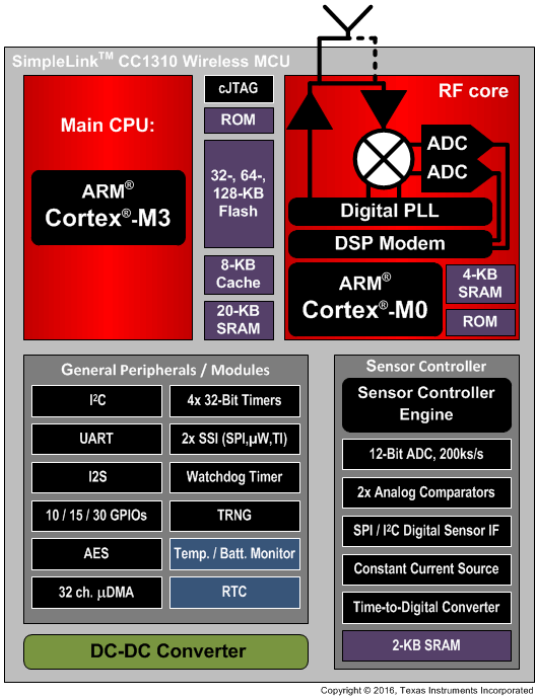


Figura 3.1: *Diagrama de bloques CC13XX.*

3.2.1.2. CC1350

El CC1350 Instruments (2016b) también es miembro de la familia CC13XX y posee, al igual que el CC1310, un transceptor de baja frecuencia, un procesador de 48 MHz Cortex-M3 y un controlador de radio dedicado (Cortex-M0). Al igual que su hermano, incorpora un modo de ultra ahorro de energía capaz de manejar sensores tanto analógicos como digitales para permitir a la *MCU* maximizar sus tiempos de reposos y así ahorrar energía. La novedad que incorpora es la capacidad de soportar tanto protocolos sub-GHz como 2,4 GHz, como por ejemplo *Bluetooth Low Energy*. Esto permite la comunicación entre redes Sub-GHz con redes 2,4 GHz, posibilitanto la utilización de soluciones que usan la banda de Sub-GHz para conseguir el mayor rango posible y la banda de 2,4GHz para, por ejemplo, realizar una conexión con un dispositivo móvil con *Bluetooth Low Energy* y así hacer posible una experiencia de usuario más rica.

Se puede ver un diagrama de bloques completo del chip en la figura 3.1. Ambas placas comparten el mismo diagrama de bloques, lo único diferente entre ambas es que la placa *CC1310* tiene varias configuraciones con 32, 64 y 128 KB de memoria *flash*, mientras que la *CC1350* tiene una única configuración con 128 KB de memoria *flash*.

3.3. Sistemas Operativos RTOS

Un *Real Time Operating System (RTOS)* es un sistema operativo diseñado para aplicaciones en tiempo real. Estos sistemas se emplean cuando hay que administrar varias tareas simultáneamente con plazos de tiempo estrictos.

Un *RTOS* se caracteriza por las siguientes características:

- Planifica las tareas en función de su criticidad.
- Predictibilidad y rapidez en el tiempo de respuesta de las tareas.
- Tienen un *kernel* interrumpible.
- Exigen pocos recursos, tanto computacionales como de memoria o almacenamiento.
- Tienen menos servicios implementados que los sistemas operativos tradicionales.

A continuación, se van a presentar dos sistemas operativos de tiempo real relevantes, el primero por ser el oficial de *Texas Instruments* y el segundo por ser el elegido para el desarrollo del proyecto.

3.3.1. TI RTOS

Texas Instruments Real Time Operating System (TI RTOS) (Instruments (2018b)) es un ecosistema de herramientas integradas creado por *Texas Instruments* para su uso en una amplia gama de sus procesadores integrados. Está compuesto por un sistema operativo de tiempo real llamado *TI RTOS Kernel* junto con componentes adicionales para conectividad de redes, administración de energía, sistema de archivos, instrumentación, comunicación entre procesadores como *DSP/BIOS Link* y *drivers* para sus dispositivos. *TI RTOS Kernel* está compuesto por una gran cantidad de componentes llamados módulos. Cada módulo provee de servicios vía *API* y son configurables individualmente.

Además, brinda soporte para varios tipos de subprocesos:

- Interrupciones *hardware* (*Hwi*).
- Interrupciones *software* (*Swi*): Estructuradas de forma similar a las interrupciones *hardware* pero que permiten que el procesamiento se posponga hasta después de qeu se haya completado una interrupción *hardware*.

- Tarea: *Thread* que puede ser ejecutado mientras se espera a que ocurra un evento.
- *Idle*: *Thread* con menor prioridad que únicamente ocurre cuando no hay otro *thread* listo para ejecutarse.

También provee un sistema de gestión de memoria que configurar el mapa de memoria del dispositivo. Además, también permite asignar y desasignar *buffers* de memoria durante la ejecución del sistema.

Por último, *TI RTOS Kernel* proporciona también módulos que permiten proporcionar información sobre cómo se está ejecutando el sistema, como por ejemplo cómo los *threads* están siendo cargado por la *CPU* o un registro de eventos a medida que ocurren tanto en la aplicación del sistema como dentro del *TI RTOS Kernel*.

3.3.2. Contiki

Contiki es un sistema operativo de código abierto diseñado específicamente para infraestructuras *Internet of Things (IoT)*. Está diseñado para operar en dispositivos con restricciones de memoria y capacidades de red, y centrado en dispositivos de bajo consumo inalámbricos como los del *IoT*. Proporciona gestión multitarea y un conjunto de protocolos de Internet (pila TCP/IP), necesitando únicamente de 10 KB de *RAM* y 30 KB de *ROM*.

Debido a que está enfocado a dispositivos *IoT* que tienen un bajo consumo energético, proporciona una serie de mecanismos para reducir el consumo energético del sistema donde es ejecutado. El mecanismo por defecto para conseguir un bajo consumo energético se llama *ContikiMAC*, con el cual los nodos pueden funcionar en modo de bajo consumo y ser capaces de recibir y retransmitir mensajes de radio. Utiliza una planificación cooperativa en el que los procesos cooperan entre sí para realzar la planificación: un proceso se mantendrá en ejecución hasta que, voluntariamente, ceda el uso de la *CPU*.

Aunque ofrece una implementación de semáforos generales, la sincronización entre procesos ocurre normalmente mediante el envío y recepción de eventos: un proceso podrá bloquearse a la espera de un evento que, normalmente, enviará otra tarea.

Para que pueda funcionar de manera eficiente y en sistemas con restricciones de memoria, *Contiki* tiene un sistema de programación basado en *protothreads*. En esencia, un *protothread* es similar a los *threads* clásicos de *POSIX*, una función que se ejecuta de forma independiente a la de otros hilos. Sin embargo, su implementación es diferente a la de los hilos tradicionales, y la planificación cooperativa permite que los *protothreads* reduzcan significativamente el *overhead* de memoria asociado a los hilos *POSIX*. El

cambio de contexto entre *protothreads* se produce mediante un mecanismo de *local continuations* que aprovecha el hecho de que se sabe en qué punto del código de un *protothread* es posible un cambio de planificación, que es en aquellas llamadas que explícitamente ceden el uso de la *CPU*.

Cuando un *protothread* cede el uso de la *CPU*, lo que realmente ocurre es que sale de la función en ejecución (mediante un *return*). Mas adelante, se volverá a ejecutar dicha función, pero mediante la inserción de construcciones *switch/case* se continuará la ejecución en la línea de código en la que se quedó anteriormente.

Los momentos en los que los *protothreads* pueden ceder voluntariamente la *CPU* son:

- Realizando una pausa de forma explícita.
- Dejando el hilo a la espera de algún evento.
- Finalizando la ejecución del hilo.

La labor de planificación de *Contiki* se realiza a nivel de procesos. Un proceso tiene dos partes, el *protothread* que representa el código que se ejecutará y el *Process Control Block* que posee información como el nombre de proceso o su estado entre otros.

Además, *Contiki* proporciona tres mecanismos de red:

- Pila *TCP/IP* que provee de conectividad *IPv4*.
- Pila *uIPv6* que proporciona conectividad *IPv6*. Además, proporciona los protocolos *RPL* para enrutado en redes *IPv6* de bajo consumo.
- Pila *Rime*: Conjunto de protocolos de red ligeros diseñados para redes inalámbricas de bajo consumo y con pérdidas, junto con el protocolo *6LoWPAN* y la capa de adaptación para IEEE 802.15.4.

  Rime es una pila de red alternativa para usar cuando no es posible el uso de pilas *IPv4* o *IPv6*. Proporciona una serie de primitivas de comunicación para sistemas inalámbricos de baja potencia. Las primitivas por defecto son: *single-hop unicast*, *single-hop broadcast*, *multi-hop unicast*, inundación de red y recopilación de datos sin dirección. Estas primitivas pueden ser usadas por si solas o junto con protocolos más complejos.

3.4. Protocolos para gestión de *beacons* BLE

Existen diferentes protocolos de emisión de datos para *beacons*, cada cual soportado por un sistema operativo determinado y que utiliza diferentes

tramas para transmitir la información deseada. Los más importantes son:

- *iBeacon*: Protocolo creado por *Apple* en 2013. Transmite un identificador único universal (*UUID*).
- *Eddystone*: Protocolo abierto creado por *Google* que pretende fomentar el Internet de las Cosas. Es similar a *iBeacon* y está soportado por dispositivos *iOS* y *Android*.
- *AltBeacon*: Protocolo desarrollado por *Radios Networks*. Define el formato de los mensajes *broadcast* por proximidad de los *beacons*.

3.4.1. Eddystone

Como se ha dicho anteriormente, *Eddystone* es un formato abierto de difusión de datos BLE para *beacons* desarrollado por *Google*. Puede ser detectado de forma nativa tanto por *iOS* como por *Android*, y por medio de diferentes APIs por cualquier otro dispositivo que tenga conexión BLE.

Eddystone define cuatro formatos de *frame* que pueden ser utilizados tanto individualmente como combinándolos según el uso que se quiera dar a los *beacons*. Estos cuatro formatos de *frame* son:

- *Eddystone UID*: Emite un identificador estático único con un formato determinado.
- *Eddystone URL*: Emite una *URL* comprimida que una vez descomprimida y parseada es directamente usable por el cliente.
- *Eddystone TLM*: Emite información del estado de los *beacons*.
- *Eddystone EID*: Componente de seguridad añadido por *Google* pensado para controlar qué clientes pueden usar las señales que emite el *beacon*.

El formato de una trama de *Eddystone* se puede ver en la tabla 3.2

Len	Type	Flags	Len	Type	Eddystone UUID	Len	Type	Eddystone UUID	Eddystone Frame
0x02	0x01	0x06	0x03	0x03	0xAA, 0xFE	0x??	0x16	0xAA, 0xFE	20 Bytes

Tabla 3.2: Formato de trama *Eddystone*.

3.4.1.1. Eddystone UID

Se trata del formato de *Eddystone* usado para la identificación de *beacons* en entornos. El *frame* está compuesto por un identificador único y estático

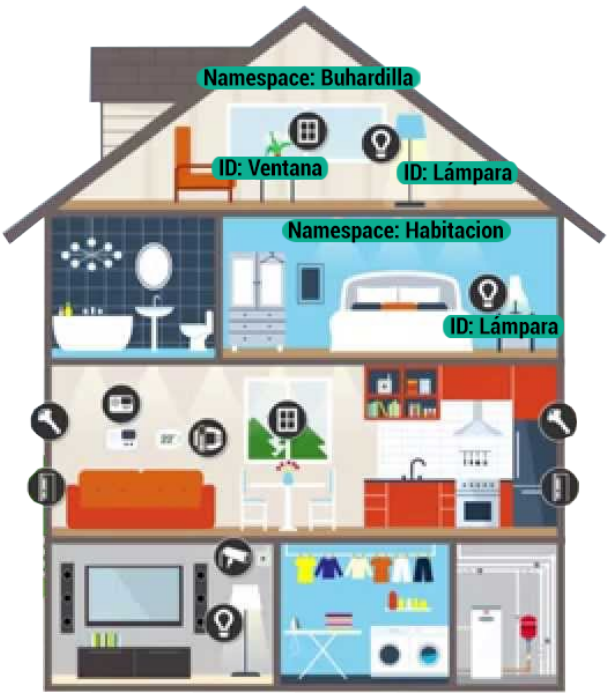


Figura 3.2: *Eddystone UID.*

formado por 16 bytes, de los cuales 10 están dedicados al *namespace* y los 6 restantes para el identificador de la instancia. Como hemos comentado anteriormente, es útil para organizar los *beacons* en grupos, identificándolos dentro de estos. Además, la división que realiza *Eddystone UID* entre el *namespace* y la instancia particular del *beacon* es útil para optimizar estrategias de escaneo *BLE*, como por ejemplo, filtrar únicamente los *beacons* por su *namespace*.

Un ejemplo de este tipo de *frame* es el que se puede ver en la figura 3.2, en la que se pueden observar cómo están organizados los *beacons* según la habitación en la que se encuentran, teniendo dentro de esta identificadores únicos para cada uno.

3.4.1.2. Eddystone URL

El formato *Eddystone URL* emite una *URL* comprimida que una vez descomprimida y parseada es directamente usable por el cliente. Para poder aprovechar el el mayor tamaño posible para la *URL* codifica tanto los prefijos de las páginas *web* como los sufijos, haciendo que ocupen únicamente un *byte*. De este modo, el prefijo *http://www.* correspondería con el valor hexadecimal *0x00*, el prefijo *https://www.* con el *0x01* y así sucesivamente, tanto para los

prefijos como para los sufijos.

Estas *URLs* pueden ser enlaces directos a páginas web, que aparecerán en las notificaciones del usuario cuando el teléfono móvil detecte la señal del *beacon*, y que al pulsarlas redirigirán a la página correspondiente.

También es posible enviar *intents* de aplicaciones *Android*, que una vez pulsadas en las notificaciones abrirán la aplicación correspondiente por la pantalla indicada en la *URL*.

Eddystone URL está directamente enfocado a una propuesta de *Google* denominada *Physical Web*. La *Physical Web* es un enfoque abierto para permitir interacciones rápidas y constantes con objetos físicos y localizaciones. Se trata de poder caminar junto a cualquier objeto, como parquímetros, juguetes, pósters, etc, o localizaciones como paradas del bus, museos o tiendas, e interactuar con ellos sin necesidad de tener ninguna aplicación instalada. La idea consiste en que cualquier objeto o lugar puede tener un *beacon BLE* asociado, que emite páginas web mediante el formato *Eddystone URL*. Esto, combinado con *Web Bluetooth*, permite crear nuevas y ricas experiencias al usuario.

Web Bluetooth es una nueva *API* experimental que permite que las páginas web se conecten directamente a dispositivos *Bluetooth* cercanos. Esta *API* permite a los dispositivos cercanos ofrecer interacciones avanzadas sin requerir una conexión a Internet. Por ejemplo, el parquímetro mencionado anteriormente podría conectarse directamente al dispositivo móvil del usuario a través de *Bluetooth*. Esta conexión directa de *Bluetooth* entre el objeto y el dispositivo móvil reduce la latencia de la red y ofrece una forma económica de permitir una interacción más extendida.

3.4.1.3. Eddystone TLM

Con *Eddystone TLM* es posible emitir información del estado de los *beacons* directamente a los clientes. Esta información, llamada telemetría, puede ser el voltaje de su batería, la humedad del dispositivo o su temperatura. Esto es útil para realizar monitorización de la salud y el estado de una flota de *beacons*.

Como *Eddytone TLM* no contiene información sobre el identificador del *beacon*, es necesario combinarlo con otro *frame* que identifique al *beacon* del que se está enviando la telemetría, como puede ser *Eddystone UID*, *Eddystone URL* o *Eddystone EID*. *Eddystone TLM* puede enviar *frames* de datos sin encriptar si se está usando junto con *Eddystone UID* o *Eddystone URL*, o encriptados si se está usando junto con *Eddystone EID* y entonces se llamará *Eddysone ETLM* (*Eddystone Encrypted TLM*).

3.4.1.4. Eddystone EID

Es un componente de seguridad añadido por *Google* al formato *Eddystone* y está pensado para poder controlar qué clientes pueden usar las señales que emite un *beacon*. Esto se lleva a cabo emitiendo un *frame* que varía con el tiempo y que puede ser resuelto mediante un identificador estable como la *Proximity Beacon API* de *Google*. El *beacon* emite un identificador efímero que cambia periódicamente. Este tiempo de cambio es determinado mediante un registro inicial que se realiza con un servicio web. El ID emitido puede ser resuelto remotamente mediante el servicio en el que ha sido registrado por aplicaciones que también están registradas en este mismo servicio. El resto de observadores únicamente verán un *frame* que varía de forma aleatoria, sin saber si se trata de un *beacon Eddystone* o de cualquier otros dispositivo *Bluetooth*.

No se recomienda intercalar el envío de este tipo de tramas con otras que emitan un identificador único del *beacon*, com pueden ser *Eddystone UID* o *Eddystone URL* ya que comprometerían la seguridad exponiendo una manera efectiva de identificar el *beacon*.

Eddystone EID soluciona los problemas de seguridad y privacidad más frecuentes que tienen los *beacons* a la hora de emitir sus señales. Al estar la señal que emite cifrada, únicamente lo usuarios que hayan sido autorizados previamente pueden entender la señal que emite y recibir la información que transmite. Por otro lado, soluciona problemas de suplantación que puedan surgir; al emitir identificadores que cambian con el tiempo, terceras personas que quieran capturar la señal y replicarla par suplantar la identidad del *beacon*, no serán capaces, ya que ésta cambia constantemente.

3.4.1.5. Provisionamiento de beacons

Los *beacons* que cumplen completamente con la especificación de *Eddystone* pueden ser provisionados para emitir cualquier trama mediante cualquier herramienta que implemente *Eddystone Generic Attribute Profile (GATT) service*. *Eddystone Generic Attribute Profile service* es un servicio de configuración que define *Eddystone* que permite la interoperabilidad entre el *hardware* de los fabricantes y los desarrolladores de aplicaciones. El *Eddystone GATT service* sirve para:

- Que los *beacons* comuniquen a las aplicaciones de sus capacidades.
- Provisionar a los *beacons* de los datos que van a emitir.
- Es necesario para la configuración de seguridad y el registro de los *beacons* de *Eddystone EID*.

Debido a que el *Eddystone GATT service* funciona mediante conexión *BLE*, el provisionamiento de los *beacons* debe hacerse en un entorno cercano físicamente a ellos para que exista esta conexión. Esto supone una gran limitación si se quieren gestionar flotas grandes de *beacons*, y más si ya se encuentran desplegados, y es una de las principales motivaciones en el desarrollo del presente proyecto.

3.5. Lightweight Machine To Machine (LWM2M)

Las aplicaciones *IoT* suelen construirse sobre redes distribuidas formadas por numerosos dispositivos de tipología y complejidad variable, por lo que es recomendable usar algún tipo de sistema de gestión de dispositivos que permita realizar acciones como:

- Gestionar las comunicaciones *Machine To Machine (M2M)*.
- Aceptar nuevos dispositivos en la red a través de un proceso de registro.
- Eliminar de forma segura dispositivos redundantes.
- Actualizar el *firmware* de dispositivos de forma remota (*Over-The-Air (OTA)*).
- Almacenar y gestionar datos procedentes de distintos dispositivos de manera individual.
- Recopilar métricas sorbe un dispositivo o grupo.
- Ejecutar un comando sobre un dispositivo o grupo.

LWM2M es un conjunto de protocolos de la *Open Mobile Alliance (OMA)* para la gestión de dispositivos y comunicaciones en entornos *M2M* e *IoT*. Su objetivo principal es la gestión de dispositivos, pero está diseñado para poder extender su funcionalidad para cubrir los requerimientos de las aplicaciones Desde el inicio se diseñó para la gestión de dispositivos con restricciones de procesamiento, memoria y alimentación, por lo que los protocolos que usa se caracterizan por un consumo de memoria restringido, comunicaciones de bajo consumo y modelos de datos minimalistas.

Se basa en el protocolo *Constrained Application Protocol (CoAP)* a nivel de conectividad y *Datagram Transport Layer Security (DTLM)* a nivel de seguridad. *LWM2M* proporciona un estándar abierto, de código libre, seguro, escalabre e interoperable para la gestión de dispositivos. Utiliza un modelo simple de recursos basados en objetos, donde cada objeto está formado por un conjunto de atributos que se pueden leer, actualizar o borrar. Además,

también permite la observación de estos y notificación cuando se produzca un cambio. Tiene soporte para *LTV*, *JSON*, texto plano y formato de datos opacos, y para *UDP*, seguridad vía *DTLS* y para servidores *bootstrap* y de gestión.

3.5.1. IPSO Smart Objects

LWM2M se basa en un modelo *Object Instance Resource (O/I/R)* donde un recurso es un elemento único y con tipo determinado. Este recurso es expuesto por un cliente *LWM2M* para ser consumido por una aplicación de gestión o control. Cada recurso tiene definidos unas características: el tipo de dato que es, su multiplicidad, es decir, si puede existir más de una instancia al mismo tiempo y las operaciones permitidas (lectura, escritura, lectura-escritura o ejecución). Los recursos se agrupan en objetos, de los cuales pueden existir varias instancias. De esta forma, el acceso a los objetos se organiza de la siguiente manera: *1000/0/1* direcciona al objeto 1000, instancia 0 y recurso 1.

Internet Protocol for the networking of Smart Objects (IPSO) es una organización que promueve la creación de un protocolo de Internet para lo que llama las "comunicaciones de objetos inteligentes". Persigue la definición de objetos inteligentes basados en el modelo de objetos y recursos *LWM2M* para proporcionar un patrón de diseño común, que asegure una alta interoperabilidad entre los dispositivos y las aplicaciones. Está construido de tal forma que no dependen del uso de *CoAP* sino que funcionan con cualquier protocolo *RESTful*. *IPSO* proporciona modelos de datos predefinidos para una gran variedad de tipos de dispositivos y que están listos para ser utilizados por cualquier aplicación que implemente *LWM2M*.

3.5.2. Eclipse Leshan

Eclipse Leshan (Foundation, 2015) es una librería que facilita la implementación de servidores y clientes *LWM2M*. Se estructura como un conjunto de librerías modulares de *Java* y tiene soporte para *IPSO Smart Objects*. Además, también tiene soporte para el registro de dispositivos usando un servidor *bootstrap*. Una vez implementado el servidor los objetos que son soportados por éste se pueden añadir mediante un fichero *XML* en el que se describen sus recursos y las características.

Por último, cabe destacar que tiene varios servidores ya implementados, y que mediante el proceso anteriormente mencionado de añadir los modelos mediante un archivo *XML*, pueden ser configurados completamente con los objetos que se necesiten, permitiendo un desarrollo rápido del servidor sin tener que implementarlo.

Capítulo 4

Desarrollo del proyecto

En este capítulo se explica cómo se ha desarrollado el proyecto: la elección del *hardware* y el sistema operativo, la topología de red desplegada, el protocolo para la capa de aplicación y la integración de todos los elementos que forman la infraestructura propuesta.

4.1. Selección de tecnologías hardware, software y de red

4.1.1. Elección del hardware

Para el desarrollo del proyecto se barajó el uso de varios *System On Chip* (véase 3.2). Finalmente se eligió desarrollar el proyecto con las placas de *Texas Instruments* de la familia CC13XX debido a la gran flexibilidad que ofrecen. Toda la familia CC13XX nos ofrece conectividad sub-GHz (véase 3.1.2) junto con el modo de energía *ultra-low power*, lo que nos permite mantener una comunicación a distancias elevadas con un consumo de energía reducido. Según el fabricante, ajustando los niveles de emisión *Rx* y *Tx* se pueden lograr comunicaciones de hasta 20 km durante 10 años con una única pila de botón. Además, la placa CC1350, nos ofrece tanto conectividad Sub-GHz como conexión *BLE*, lo que nos permite utilizarla como *beacon* mientras se mantiene esta comunicación mediante Sub-GHz.

En el desarrollo del proyecto se han usado los dos formatos de placa que ofrece *Texas Instruments* para las CC1350: en formato placa de desarrollo con la *SimpleLink Dual-Band LAUNCHXL-CC1350 Wireless MCU LaunchPad Development Kit* (a partir de ahora llamada únicamente *CC1350 LaunchPad* o *LaunchPad*), como en formato de *kit* de demostración con la *SimpleLink CC1350 SensorTag* (llamado a partir de ahora *CC1350 SensorTag* o *SensorTag*).

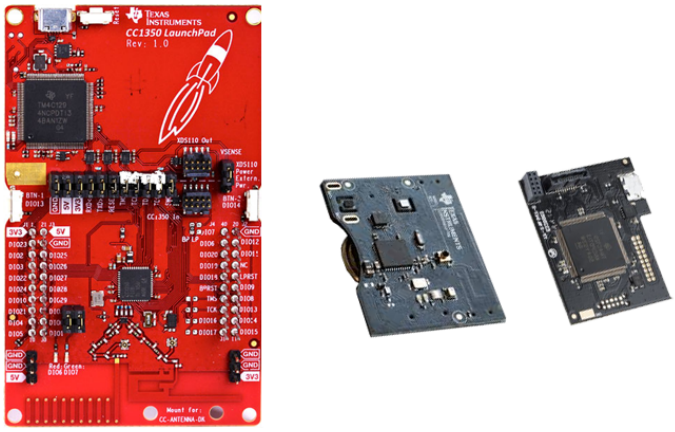


Figura 4.1: *Launchad*, *SensorTag* y *DevPack*.

La ventaja que ofrece la *CC1350 LaunchPad* es la posibilidad de incluir de forma sencilla sensores externos a la placa según las necesidades del proyecto, además cuenta con *debugger* incorporado dentro de la propia placa. Por el contrario, la *CC1350 SensorTag* incluye incorporados varios sensores (humedad, temperaturas y giroscopio entre otros), lo que la hace idónea para un desarrollo de aplicaciones más rápido. Como punto negativo, la *CC1350 SensorTag* no incluye *debugger* incorporado en la placa, sino que se debe utilizar el *debugger* externo de *Texas Instruments* llamado *SimpleLink SensorTag Debugger DevPack* (Instruments, 2018a) (que a partir de ahora se denominará únicamente *Debugger DevPack* o *DevPack*).

Aunque la placa en formato *SensorTag* ya tiene sensores incorporados, es posible incluir sensores externos mediante el uso del *DevPack*, ya que incluye conexiones a las que se pueden soldar estos sensores, aunque es más complejo realizarlo que en la *LaunchPad* donde se puede realizar la conexión de estos sensores sin necesidad de soldar nada.

La placa CC1310, que únicamente dispone de conectividad sub-GHz, se ha usado para el *router* que hace la función de *gateway* en el sistema (capítulo 4.4) y se ha elegido el formato *LaunchPad* ya que era necesario incluirle un módulo *Ethernet* externo.

Una imagen de ambas placas y del *DevPack* se puede ver en la figura 4.1 (las imágenes no se encuentran escaladas, la placa en formato *LaunchPad* es bastante más grande que en formato *SensorTag*).

4.1.2. Elección del sistema operativo

Para la elección de sistema operativo de tiempo real se barajaron dos opciones: utilizar el propio sistema operativo que ofrece *Texas Instruments* llamado *TI RTOS* (capítulo 3.3.1) o utilizar el sistema operativo *Contiki* (capítulo 3.3.2) que es compatible con las placas seleccionadas.

TI RTOS nos ofrece una gran variedad de funcionalidades y facilidades para controlar las placas. Provee módulos para la gestión de sensores externos y una pila de red. Por otro lado *Contiki* ofrece igualmente funcionalidades para la gestión de sensores externos así como varias pilas de red. Finalmente, se decidió usar la última opción ya que además de ofrecer las funcionalidades de *TI RTOS*, ofrece un gran soporte para aplicaciones como por ejemplo *DHCP*, *MQTT*, *CoAP* o *LWM2M*, entre otras.

Para poder realizar un desarrollo más flexible se definió una nueva jerarquía de proyecto distinta a la que sugiere *Contiki* que se explicará más adelante en la sección 4.2.

4.1.3. Topología de red

Debido a que los *beacons* se iban a desplegar en un entorno no controlado en el que puede haber pérdida de conectividad entre ellos, era necesario elegir una topología de red que fuera resistente a caídas.

Por estos motivos se eligió una distribución de red en malla, la cual permite que los nodos se comuniquen entre sí, independientemente del punto de acceso.

En este tipo de red los dispositivos que actúan como nodos pueden no mandar directamente sus paquetes al punto de acceso, sino que pueden pasárselos a otros nodos para que lleguen a su destino. Con esto se consigue que si un nodo falla no se rompa toda la red, como los nodos están conectados a diferentes nodos hace más fácil que la transmisión de datos siga activa aun habiéndose producido una caída.

Además, como los *beacons* se iban a controlar de forma remota mediante el uso de un servidor, era necesario que existiera una conectividad punto a punto entre éste y los nodos desplegados.

Para esto existen diferentes protocolos de la capa de enlace de datos como *LoRa*, *ZigBee* o *6LoWPAN*.

Finalmente se eligió *6LoWPAN* ya que está completamente soportado por *Contiki* y nos ofrece conectividad punto a punto mediante *IPv6*.

IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) es un estándar que posibilita el uso de *IPv6* sobre redes basadas en el estándar *IEEE 805.15.4*. Hace posible que los nodos de una red inalámbrica se

IP		6LoWPAN
HTTP, FTP, DHCP...	Aplicación	CoAP, MQTT, XMPP...
TCP | UDP | ICMP	Transporte	UDP | ICMPv6
IP	Red	IPv6
Ethernet MAC	Enlace de datos	LoWPAN
		IEEE 802.15.4 MAC
Ethernet PHY		IEEE 802.15.4 PHY

Tabla 4.1: Comparación pila IP y pila *6LoWPAN*.

conecten directamente con otros dispositivos mediante *IP*.

Este estándar actúa como una capa de adaptación entre la tecnología estándar *IPv6* y el medio ofrecido por el *IEEE 802.15.4* para comunicaciones con pérdidas y de bajo consumo.

6LoWPAN deriva de *LoWPAN*, que es una instancia particular de *LNN* formada con dispositivos compatibles con el IEEE 802.15.4. Las características de *LoWPAN* son:

- Apto para sistemas con capacidad de proceso y memoria limitadas.
- Bajo consumo energético.
- Corto alcance.
- Bajo coste.

Como protocolo de enrutamiento se ha hecho uso de *RPL*, que es un protocolo diseñado específicamente para redes de bajo consumo y con pérdidas.

RPL es un protocolo de vector de distancias proactivo. En éstos cada nodo conoce a los vecinos conectados a él y los costes de dichos enlaces. Cada cierto tiempo el nodo transmite su tabla de enrutamiento a sus vecinos y éstos recalculan sus tablas si existe una nueva información. En estos protocolos los nodos no conocen toda la topología de la red, únicamente la de sus nodos vecinos.

En la tabla 4.1 se puede ver un esquema de la pila de *6LoWPAN* en comparación con la pila *IP*.

4.1.4. Protocolo capa de aplicación

Para la gestión remota de los *beacons* se consideraron diferentes protocolos de aplicación. Finalmente se decidió elegir *LWM2M*, ya que nos brinda más flexibilidad y potencia respecto de otros protocolos. Además, *Contiki*

cuenta, dentro de sus aplicaciones, con una implementación de *LWM2M* y de *IPSO Smart Objects*.

Se barajaron dos opciones para el servidor de *LWM2M*, una basada en *C* llamada *Eclipse Wakaama* y otra en *Java* llamada *Eclipse Leshan*. *Eclipse Wakaama* no llega a ser una librería en sí, sino que son un conjunto de archivos que deben compilarse con la aplicación que se desarrolle. Como ya se ha dicho, está escrito en *C*, es mono-hilo y está diseñada para ser portable a sistemas *POSIX*. Puede funcionar tanto en modo servidor como en modo cliente. En el otro lado tenemos a *Eclipse Leshan*, que es una librería para facilitar la implementación de servidores y clientes *LWM2M*. Se estructura como un conjunto de librerías modulares de *Java* y además tiene soporte para *IPSO Smart Objects*.

Finalmente se eligió utilizar *Eclipse Leshan* debido a las características descritas en el párrafo anterior: su estructuración como librerías modulares y el soporte de los *IPSO Smart Objects*. Además, *Eclipse Leshan* ofrece un servidor de *LWM2M* ya implementado donde se pueden añadir los *IPSO Smart Objects* que se deseen mediante el uso de modelos que se definen en un archivo *XML*, lo que facilita el trabajo al no tener que implementar el servidor desde cero pero a la vez poder personalizarlo con los *IPSO Smart Objects* que se quieran utilizar.

Así, la topología de red que se desplegó finalmente sería la reflejada en la imagen 4.2, donde se tiene el servidor *LWM2M* con *Eclipse Leshan*, el *router IP64* (que se hablará del él en la sección 4.4) como *gateway* y los nodos como clientes *LWM2M* y emitiendo mediante *BLE* señales *broadcast*.

4.2. Organización del proyecto

Para organizar el proyecto y poder hacer más fácil la programación y compilación para *Contiki* se llevó a cabo la estructuración del proyecto que se detalla a continuación (todas las rutas son desde la capeta padre del proyecto):

- *contiki/:* *Fork* del repositorio oficial de *Contiki* con las modificaciones pertinentes (sección 4.3).
- *Debug/:* Carpeta donde se generan los binarios y archivos intermedios de compilación (*.obj*) para el modo de compilación *debug*. Se crean carpetas dentro de ésta para cada proyecto y cada plataforma. Si se quiere compilar varios proyectos distintos para una misma plataforma es recomendable realizar un *make clean* entre ellos.
  Por ejemplo, para un proyecto llamado *project1* compilado para las plataformas *everis-iot* y *srf06-cc26xx* quedaría de la siguiente manera:

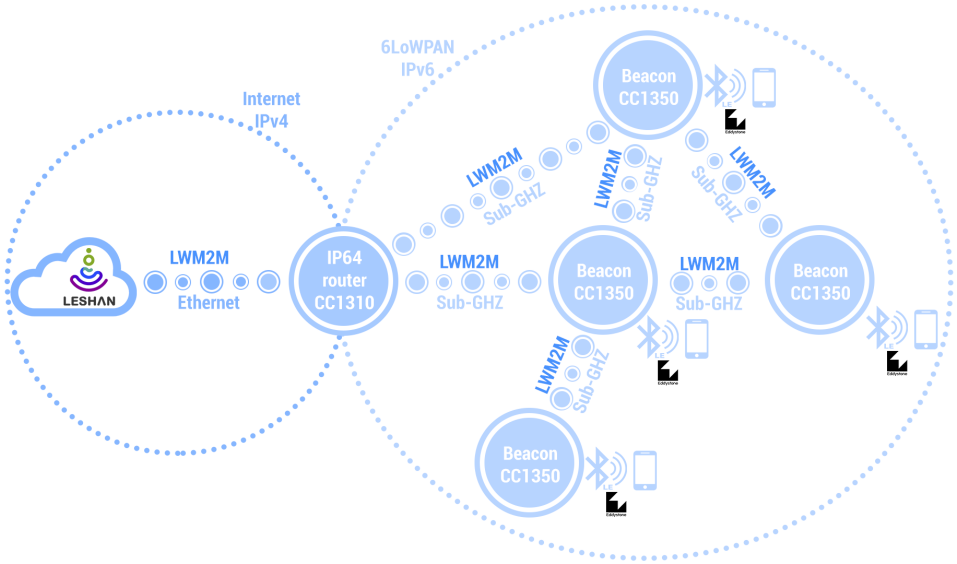


Figura 4.2: Topología de red desplegada.

- *obj-everis-iot/:* Archivos intermedios para la plataforma *everis-iot*.
- *obj-srf06-cc26xx/:* Archivos intermedios para la plataforma *sr06-cc26xx*.
- *project1-everis-iot/:* Archivos binarios finales del proyecto *project1* para la plataforma *everis-iot*.
- *project1-srf06-cc26xx/:* Archivos binarios finales del proyecto *project1* para la plataforma *srf06-cc26xx*.

- *doc/:* Para la documentación del proyecto.
- *Release/:* Capeta donde se generan los binarios y objetos intermedios de compilación (*.obj*) para el modo de compilación *release*. Está estructurada del mismo modo que la capeta *Debug* mencionada anteriormente.
- *src/:* Fuentes del proyecto.
  - *ip64-router/: Router IPv4-IPv6* para traducción de paquetes entre redes con distinta versión de *IP* (sección 4.4).
  - *lwm2m/:* Cliente *LWM2M* basado en *Contiki* (se hablará mas de la implementación en la sección 4.3.2).
  - *sensniff/: Sniffer* para monitorizar la red *mesh*. Se ha usado para poder tener otra forma de depurar fallos a la hora de desarrollar

el proyecto y poder ver en tiempo real qué paquetes se enviaban por la red.
Está basado en el *sniffer* incluido en los ejemplos de *Contiki* y se usa junto con *sensniff* (*enlace al proyecto de GitHub*). Para su uso se programa un dispositivo con el código y junto con *sensniff* corriendo en un *host* es posible ver el tráfico de red en *Wireshark*.

- *Makefile: Makefile* principal del proyecto (en la sección 4.3.4 se hace referencia al sistema de *Makefiles* utilizado).

4.3. Modificaciones en Contiki

Para desarrollar el proyecto era necesario realizar diferentes modificaciones a *Contiki*. Se creó una plataforma propia basada en la *srf06-cc26xx*, se modificó parte del comportamiento de *LWM2M* para añadir alguna funcionalidad necesaria para el proyecto, se crearon nuevos *IPSO Smart Objects* y se modificaron los *Makefiles* de *Contiki* para que se adaptarán a la nueva estructura de proyecto explicada en la sección anterior.

Además, se programó un *router* capaz de traducir direcciones *IPv6* a *IPv4* y viceversa (sección 4.4) y se modificó parte del funcionamiento del *rf-core* de las placas *CC26XX-CC13XX* para adaptar el *broadcast* que realizaban mediante *BLE* al formato *Eddystone URL* (sección 4.5)

4.3.1. Plataforma everis-iot

Para desarrollar el proyecto se creó una nueva plataforma basada en la existente en *Contiki* llamada *srf06-cc26xx* que da soporte a las placas de *Texas Instruments* de la familia *CC13XX* y *CC26XX*.

En esta plataforma se han añadido archivos para permitir la traslación de paquetes entre redes *IPv4* e *IPv6*, los cuales se explican en la sección 4.4.

También se han realizado modificaciones y añadido nuevos archivos para la compatibilidad de los *IPSO Smart Objects* que se han creado y para añadir nuevas funcionalidades a *LWM2M*.

Además, se ha modificado los archivos *board.h* de las placas para adaptarlos a las modificaciones que se han realizado. Por ejemplo, se han añadido definiciones para poder saber si una placa tiene sensor de temperatura o si puede funcionar como *beacon*.

Algunos de los cambios que se han comentado se desarrollarán más ampliamente en secciones posteriores.

4.3.2. LWM2M

A nivel de *LWM2M*, se han realizado varias modificaciones para adaptarlo a nuestro proyecto. Se ha modificado el motor de *LWM2M* para soportar actualizaciones del registro. Cuando un dispositivo se registra en el servidor establece un tiempo de vida (lt) durante el cual el registro es válido. Si éste tiempo expira y el servidor no ha recibido ningún paquete del dispositivo, lo elimina de la lista de dispositivos registrados.

El dispositivo debe realizar una actualización de registro antes de sobrepasar este tiempo de vida (lt). Cuando se envía una actualización de registro el tiempo de vida (lt) se reinicia, haciendo que el servidor no elimine de la lista de dispositivos registrados el dispositivo en cuestión. Esta actualización se realiza enviando un mensaje *CoAP* con el *location_path* que el servidor le ha asignado al registrarse. Se puede ver un diagrama de secuencia de cómo se realiza ésto en la figura 4.3.

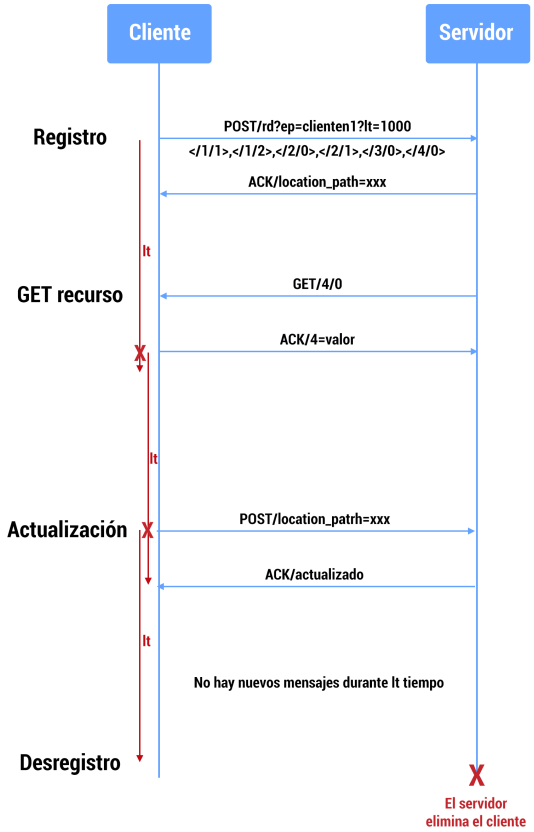


Figura 4.3: Diagrama de flujo *LWM2M*.

También se han desarrollado otras pequeñas modificaciones como aumen-

tar el tamaño del *buffer* destinado al *endpoint*, ya que al usarse los primeros bytes de la dirección *MAC*, si los dispositivos comienzan por la misma dirección al tener el mismo fabricante, no sería posible distinguirlos entre ellos. Por esto vemos necesario utilizar la dirección *MAC* completa como identificador del dispositivo.

4.3.3. IPSO Smart Objects

Para definir los diferentes sensores y actuadores que queríamos controlar mediante *LWM2M* hemos usado *IPSO Smart Objects*. Se han creado los objetos para la humedad, la temperatura, el control del *buzzer* del *SensorTag* y la conectividad. Para la temperatura y la humedad se ha creado un único archivo para el *SensorTag* que controla ambas, ya que el sensor que tomar medidas de temperatura y humedad es el mismo. También se ha creado otro objeto para la gestión de los *beacons* que se explicará mas adelante en la sección 4.5.

Los *IPSO Smart Objects* en *Contiki* constan de dos partes: el *ipso-object* localizado en la carpeta *contiki/apps/ipso-object* y el *driver* localizado en la carpeta de la plataforma correspondiente. El primer archivo, el *ipso-object*, se encarga de gestionar todo lo relacionado con *LWM2M*. El registro del objeto y de sus recursos en el servidor *LWM2M* y las lecturas o modificaciones de estos recursos. El segundo, el *driver*, es el encargado de calcular los valores que necesita el primer archivo para poder dar respuesta al servidor. Por ejemplo, en nuestro objeto *ipso-connectivity* hay operaciones para poder leer el *RSSI* de la conexión entre un nodo y su nodo padre. Cuando el servidor pida al dispositivo que le de su *RSSI*, nuestro *ipso-connectivity* responderá con la última medida que tenga almacenada.

El *RSSI* lo calcula el *ipso-connectivity* invocando periódicamente al *ipso-connectivity-driver*. Éste se encuentra en la carpeta de la plataforma *everis-iot* y ejecuta un proceso que realiza un *ping* a su nodo padre, y con el *ACK* que recibe calcula el *RSSI* que lee el *ipso-connectivity*. Para que *Contiki* sepa qué *drivers* debe usar se tienen que definir en el *project-conf.h* con el nombre del *driver* que debe estar en la carpeta de la plataforma.

Hay algunos *IPSO Smart Objects* que no necesitan de *driver* para funcionar, ya que pueden calcular sus valores directamente desde el objeto *ipso-object*, es el caso por ejemplo del *ipso-beacon-control* que realiza la gestión de los *beacons* desde el mismo archivo. En cambio, el objeto *ipso-temperature-humidity* sí necesita de este *driver* para funcionar ya que la lectura de los sensores en *Contiki* se realiza de forma asíncrona, por lo que necesita de otro proceso que se encargue de realizar esta lectura mientras que él se encarga de gestionar las peticiones de *LWM2M*.

4.3.4. Makefile

Para adaptar *Contiki* a la nueva estructura de proyecto creada, fue necesario modificar el sistema de *Makefile* que usa *Contiki* por defecto, ya que con ésta los ficheros fuente deben estar en la capeta *examples* dentro de *Contiki*. La jerarquía final de *Makefiles* que se ha creado ha sido la siguiente (todas las rutas son desde la capeta padre del proyecto). Los *Makefiles* se describen en orden que van siendo invocados:

- *Makefile: Makefile* principal del proyecto. En éste se especifica la configuración general de *Contiki* y se añaden las aplicaciones que se van a usar.
  De debe definir si se quiere compilar en modo *debug* o *release* y qué proyecto se quiere compilar. Esto se realiza de la siguiente manera: *make PROJECT-NAME [DEBUG=0/1]*. Si no se especifica el proyecto que se quiere compilar fallará la orden *make*. Si no se especifica si se quiere compilar para el modo *debug* o *release*, se asumirá que *DEBUG=1*.

- *src/project-name/Makefile.project: Makfile* definido para cada proyecto donde se especifican propiedades específicas de cada uno.

- *contiki/Makefile.include: Makefile* principal de *Contiki* donde las carpetas para los objetos intermedios y binarios son creadas y los archivos fuentes de *Contiki* son compilados.

- *contiki/platform/platform-name/Makefile.platform: Makefile* específico para cada plataforma donde se especifican las propiedades de cada una y los archivos fuentes de éstas son añadidos a la compilación.

- *contiki/platform/platform-name/board-name/Makefile.board-name: Makefile* de la placa de cada plataforma donde se especifican las propiedades de cada una y se añaden los archivos fuentes de éstas a la compilación.

- *contiki/cpu/cpu-name/Makefile.cpu-name: Makefile* específico de cada *CPU* donde el compilador para cada una es definido e invocado. Aquí es donde se establecen la mayoría de *flags* para la compilación y donde se crean los objetos finales.
  Algunos *Makefiles* de las *CPU* invocan a su vez a otros *Makefiles*, por ejemplo el *Makefile* de la *cc26xx-cc13xx* llama al *Makefile* de *ARM* al estar basada su estructura en este procesador.

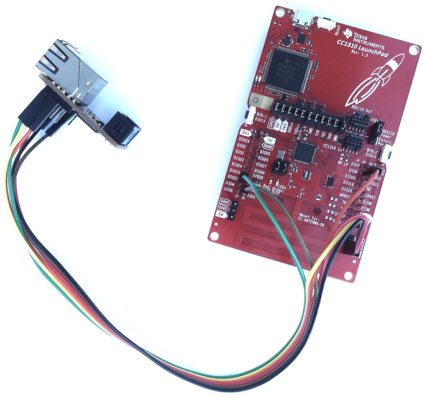


Figura 4.4: *Router IP64* con el módulo *ENC28J60-H* y la placa *TI CC1310 LaunchPad*.

4.4. Router IP64

Para que hubiera una conectividad punto a punto en el sistema, era preciso que el *gateway* que se utilizase pudiera enlazar redes *IPv4* e *IPv6* de forma transparente. Para ésto, es necesario que el *gateway* tenga dos interfaces de red, una *IPv4* para conectar al exterior de la red *mesh* y otra *IPv6* para la comunicación con ésta. Además, es necesario que realice el *forwarding* de paquetes entre ambas redes, para lo que es necesario un método de traducción de direcciones *IPv4* a *IPv6* y viceversa.

Para el *router*, como se ha mencionado en la sección 4.1.1, se ha utilizado la placa *CC1310* de *Texas Instruments* en su versión *LaunchPad*, la cuál equipa una conexión por medio de tecnología sub-GHz, lo que resuelve uno de los requisitos, que es la conectividad *IPv6* con la red *mesh*. A la placa se le ha acoplado un módulo *Ethernet ENC28J60-H* de *Molinex* que funciona por medio de *SPI*, lo que nos permite disponer de una interfaz de red *IPv4* para la comunicación con el exterior. La conexión del módulo *Ethernet* a la placa se puede ver en la figura 4.4.

El código para hacer funcionar el *router IP64* en *Contiki* consta de varias partes:

- Código principal del *router*.
- Configuración de red del *router* para la plataforma *everis-iot*.
- *Driver* para la comunicación entre la placa y el adaptador de red.

La configuración de red del *router* para la plataforma *everis-iot* se define en el archivo *ip64-conf.h*. En éste se especifica que las interfaces de entrada

y de *fallback* deben ser de tipo *Ethernet IP64*, que se debe usar un servidor *DHCP* para configurar la *IP* de ésta interfaz y el *driver* del módulo *Ethernet* que se está usando.

La interfaz *fallback* comentada en el párrafo anterior, es la interfaz que se usará para redirigir los paquetes que no sean enrutables dentro de la red *mesh*.

El *driver* para la comunicación entre la placa y el adaptador de red está definido en varios puntos. En la carpeta */dev/enc28j60* hay varios archivos que se encargan de la comunicación con el módulo *ENC28J60* y que vienen ya definidos en *Contiki*. Éstos archivos son los encargados de escribir y leer de los registros de memoria del módulo *Ethernet* para controlar el envío y recepción de paquetes, y usan varias funciones que dependen de la placa que se esté usando.

Estas funciones que dependen de cada placa son las encargadas de escribir por medio de *SPI* en los registros del módulo, y dependiendo de la placa que se utilice, se realizará de una manera o de otra. Para definirlo es necesario hacerlo en la carpeta de la plataforma correspondiente, en el archivo *enc28j60-arch-gpio.c*. En éste se definen los puertos de la placa que se utilizarán para la comunicación con el módulo *Ethernet* y se realiza la lectura y escritura de estos puertos por medio de *SPI*.

El código principal del *router* nada más comenzar se coloca como nodo raíz de la red *RPL*, posteriormente inicializa el módulo *IP64* para la traducción de paquetes, apaga el ciclo de radio *Rx* para dejar el transmisor de radio encendido en todo momento y no perder ningún paquete, y por último espera a recibir algún evento por la interfaz de red o de radio para realizar la traducción de direcciones y el *forwarding* de los paquetes.

El módulo *IP64* es el encargado de inicializar el *driver IP64* definido en el archivo de configuración de red. Posteriormente inicializa el el cliente *DHCP* que pide una dirección *IPv4* al servidor *DHCP* de la red donde se encuentra conectado por medio de *Ethernet*. Además, en éste también se encuentran las funciones encargadas de la traducción de direcciones *IPv4* a *IPv6* y viceversa.

4.5. Implementación de beacons en Contiki

Para la gestión de *beacons* en *Contiki* se ha creado un nuevo *IPSO Smart Object* para poder controlar el *advertising* que se realiza de forma remota mediante *LWM2M*.

Éste objeto cuenta de tres recursos, uno para controlar el estado del *beacon* (si está emitiendo o no), otro para el mensaje (lo que emite) y un

último para leer el *PDU* del *beacon*, es decir, el número de mensajes emitidos.

Este recurso puede emitir tanto mensajes con formato *Eddystone* como mensajes sin formato, y se controla mediante la macro *EDDYSTONE_ENABLED*.

A la hora de escribir un mensaje mediante *LWM2M*, esta clase controla que el mensaje sea válido. En el caso de que no se quiera usar el formato *Eddystone*, simplemente se comprueba que la longitud del mensaje no supere el máximo de caracteres permitidos para su emisión.

En cambio, si se ha seleccionado el formato *Eddystone* para emitir datos, la *URL* introducida se valida comprobar si es correcta. Para realizar esta validación se comprueba que tiene un prefijo y sufijo válido, y que además, no supera la cantidad de *bytes* permitidos para la emisión.

En ambos casos, si el mensaje o *URL* introducido es válido, se invocará a la función *rf_ble_beacond_config* con el intervalo deseado entre las ráfagas de emisión de datos. En el caso de que el mensaje mensaje se envíe sin formato, se llamará a la función con un argumento del tipo *const char**, y en el caso de que se quiera emitir con formato *Eddystone URL*, la *URL* se pasará a la función como un *uint8_t**.

4.5.1. Eddystone URL en Contiki

Para poder adaptar el formato de *Eddystone* en *Contiki* se han tenido que realizar modificaciones en el archivo *rf-ble* que se encuentra dentro de los archivos de la *CPU* para las placas de *Texas Instruments* de la familia *CC13XX* y *CC26XX*.

En primer lugar, se ha redefinido la función mencionada anteriormente, *rf_ble_beacond_config*, para que acepte un tipo de argumento u otro según se esté utilizando *Eddystone* o no. Si la macro *EDDYSTONE_ENABLED* no está definida usará la función por defecto para estas placas, si en cambio sí está definida, utilizará la segunda función.

Además, siguiendo con la lógica anterior, la variable de este archivo encargada de almacenar la información de emisión del *beacon* también ha sido modificada, teniendo una variable del tipo *char[]* si no está activado el formato *Eddystone URL* y del tipo *uint8_t[]* si sí lo está.

El mayor cambio viene en el proceso encargado de la emisión de ráfagas de datos. Si no se está usando el formato *Eddystone*, la cabecera de la trama que se emitirá será la que tiene *BLE* por defecto y en el campo destinado al mensaje de *advertising* será donde se escribirá el mensaje anteriormente configurado. Si en cambio el *beacons* sí está configurado para emitir tramas en formato *Eddystone*, la cabecera de la *trama*que se emitirá no será configurada con la configuración por defecto de *BLE*, sino que se rellenará con el formato

Len	Type	Flags	Len	Type	Eddystone UUID	Len	Type	Eddystone UUID	Frame Type	TX Power	URL Scheme	Encoded URL
0x02	0x01	0x06	0x03	0x03	0xAA, 0xFE	0x??	0x16	0xAA, 0xFE	0x10	0x??	0x??	17 Bytes

Tabla 4.2: Formato trama *Eddystone URL*.

de trama correspondiente a *Eddystone URL*, que se puede ver en la tabla 4.2.

Para realizar la codificación de los prefijos y los sufijos, se han definido dos *arrays* en los que se guardan las posibles opciones. Para reconocer qué codificación se debe utilizar, se recorre el *buffer* donde se almacena la *URL* sin codificar con un bucle, y si coincide se añade la posición del *array* del prefijo o sufijo correspondiente a la *URL* codificada. Al final del proceso de codificación, se tiene en un *buffer* la *URL* codificada, con los prefijos correspondientes, y los sufijos si los hubiera.

Para entender un poco más este proceso de codificación, se va a poner un ejemplo de cómo se codificaría una *URL*, como podría ser *https://google.com*. En primer lugar se recorrería la *URL* buscando un prefijo válido, en este caso sería *https://www.*, que corresponde con la codificación 0x01. Posteriosmente se leería la *URL* hasta llegar al sufijo, *.com*, que corresponde con la codificación 0x07. Así, al final la *URL https://www.google.com* quedaría como *01google07*. Si la *URL* tuviera más caractecteres tras el sufijo se añadirían después, así *https://www.google.com/S6zT6P* quedaria de la siguiente manera: *01google00S6zT6P*. Si por el contrario no tuviera ninguna terminación válida, como por ejemplo *https://goo.gl/S6zT6P*, no se añadiría ningún sufijo y quedaría de la siguiente manera: *03goo.gl/S6zT6P*. Se puede ver la codificación de todos los prefijos y sufijos aceptados por *Eddystone URL* en la tabla 4.3. Como se puede ver, las codificaciones del rango 0x07-0x13 son las mismas que las del rango 0x00-0x06 pero sin "\".

Este *buffer* con la *URL* codificada se concatena al final de la trama *BLE* que se tenía preparada con la cabecera de *Eddystone URL* y se emite mediante la interfaz *BLE* de la placa.

Consideramos que los cambios que se han implementado son bastante importantes, ya que dan mayor funcionalidad a *Contiki* al dotarle de algún protocolo para hacer *broadcasting* con *beacons*, algo que hasta ahora únicamente podía hacerse sin seguir ningún protocolo emitiendo tramas *BLE* sin un formato determinado. Por esto se ha decidido realizar una *pull request* al repositorio oficial de *GitHub* de *Contiki* para que se incluya en la distribución oficial.

Prefijos

Hex	Prefijo
0x00	http://www.
0x01	https://www.
0x02	http://
0x03	https://

Sufijos

Hex	Sufijo	Hex	Sufijo
0x00	.com/	0x08	.org
0x01	.org/	0x09	.edu
0x02	.edu/	0x0A	.net
0x03	.net/	0x0B	.info
0x04	.info/	0x0C	.biz
0x05	.biz/	0x0D	.gov
0x06	.gov/	0x0E...0x20	Reservado para usos futuros
0x07	.com	0x7F...0xFF	Reservado para usos futuros

Tabla 4.3: Codificación de prefijos y sufijos en *Eddystone URL*.

4.6. Resultado del desarrollo y caso de uso

Para obtener una visión general del resultado global del trabajo realizado, a continuación se describe un caso de uso simplificado pero real mediante el cual se ilustra el funcionamiento general del sistema. En este caso, se conectarán los dispositivos *hardware*, se provisionará un *beacon* desde *Leshan* y se verá cómo se notifica la *URL* en el teléfono móvil una vez ha sido emitida por el *beacon*.

En primer lugar es necesario conectar el *router IP64* a la corriente para dotarlo de alimentación y a una roseta de un *router* mediante *Ethernet* para que adquiera una *IP* en la red. Transcurridos unos instantes, cuando el servidor *DHCP* le asigne una *IP* válida, el *router IP64* estará listo para redirigir paquetes de la red mallada fuera de ésta (hacia Internet) y viceversa.

Una vez desplegado el *router IP64*, bastaría con alimentar un nodo y éste detectaría automáticamente la red mallada si se encuentra dentro de su alcance y se uniría a ella. En caso de que no detectase ninguna red disponible se quedaría a la espera de unirse a una de forma indefinida. Tras el proceso de unión a la red, enviaría un paquete al servidor *Leshan* con los *IPSO Smart Objects* de los que dispone para su registro. Una vez realizado este registro y habiendo respondido el servidor con el *location_path* que le ha asignado, el nodo estaría listo para controlarse mediante el servidor *LWM2M*.

Nada más abrir la página web donde se encuentra alojado el servidor *LWM2M* aparecerá la lista de nodos actualmente registrados. Esto se puede observar en la figura 4.6a, donde aparece listado el nodo que acabamos de conectar. Al pulsar sobre él aparecerían todos los objetos que tiene registrados y sobre los cuales se pueden realizar las operaciones de lectura o escritura deseadas.

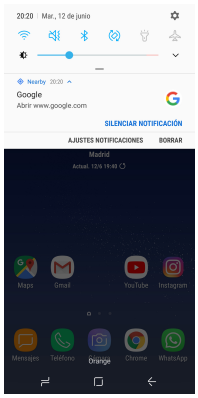
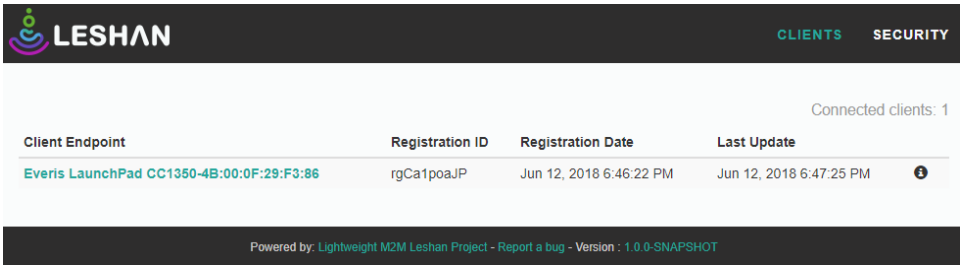


Figura 4.5: Notificación *Nearby* con la *URL* de un *beacon* cercano.

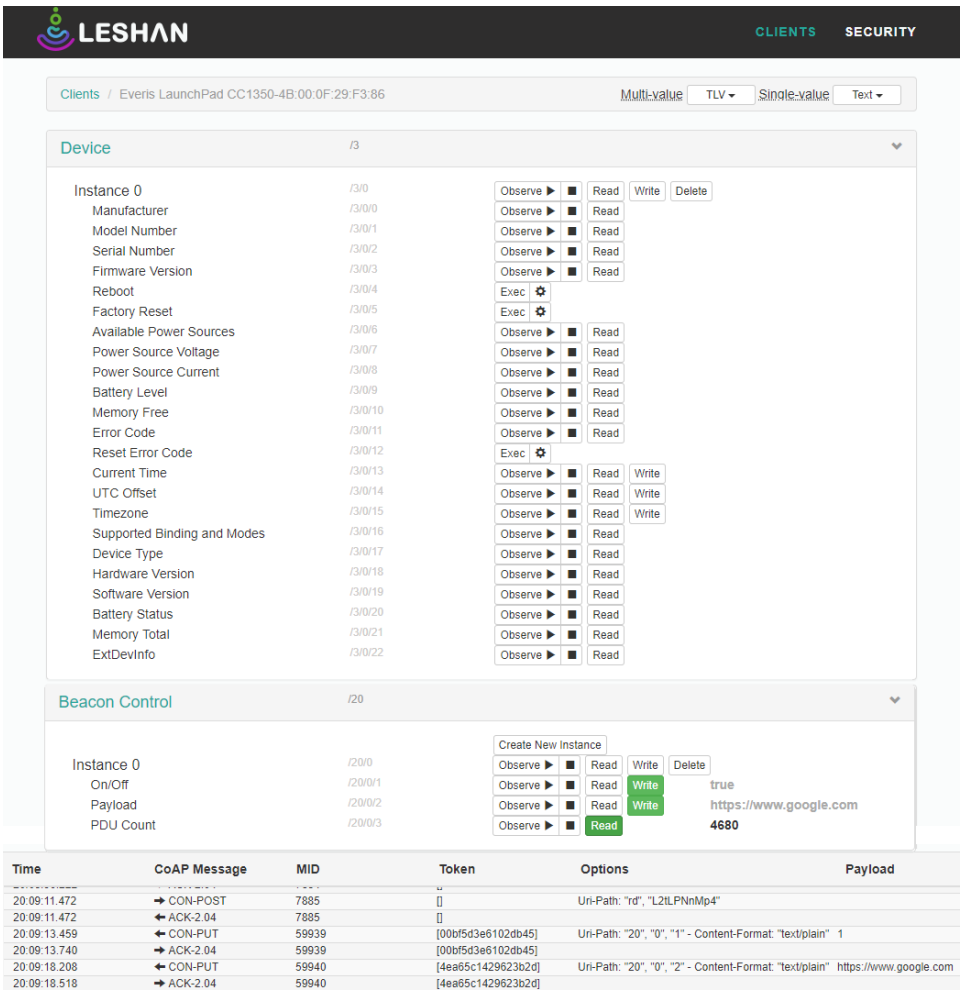
Para controlar los *beacons* se deberá usar el *IPSO Smart Object* definido para tal fin, que en nuestro caso es el llamado *Beacon Control*. Este objeto tiene tres recursos definidos: (1) encender o apagar la emisión del *beacon*, (2) leer o escribir su *payload* y (3) leer el *PDU count*, es decir, el número de mensajes emitidos. Para configurar el *beacon* es necesario escribir su *payload* primero y posteriormente encenderlo, ya que si se intenta encender el *beacon* sin haber configurado antes su *payload* la operación no se realizará con éxito (al no haber ningún dato a emitir). Si una vez que se ha configurado el *beacon* se quiere modificar el *payload* que está emitiendo, puede realizarse sin necesidad de un reinicio.

En la figura 4.6b se puede observar cómo se ve un nodo en el servidor *Leshan* y el objeto utilizado para controlar el *beacon* configurado con el *payload* *"https://www.google.com"*, encendido y con la cuenta de los mensajes que se ha enviado. Además en esta misma figura se puede observar una característica que no se ha introducido anteriormente: el servidor *Leshan* permite ver el intercambio de mensajes *CoAP* que ha habido entre el servidor y el nodo. Así puede verse (de abajo a arriba) que se ha escrito en el dispositivo con la *URL* que se quería emitir, que se ha escrito un "1" en el recurso *"on/off"* para encender el *beacon* y que empiece a emitir, y que el dispositivo ha realizado un *POST* al servidor con el *location_path* que le ha asignado para preservar la conexión.

Una vez introducida la *URL* que se desea emitir y encendido el *beacon*, éste empezará a emitir tramas con dicha *URL*. Si nos hallamos cerca del *beacon*, y habilitamos el servicio de ubicación y *Bluetooth* del teléfono móvil, nos aparecerá una notificación de *Nearby* en el panel de notificaciones con la *URL* que está emitiendo el *beacon*; tras pulsarla, nos redirigirá a la página correspondiente. Se puede observar cómo aparece la notificación del *beacon* configurado anteriormente en la figura 4.5.



(a) Lista de nodos.



(b) Vista de un nodo y del objeto para el control del *beacon*.

Figura 4.6: Lista de nodos y vista de un nodo controlando un *beacon* en *Leshan*.

Capítulo 5

Aplicaciones

El trabajo desarrollado durante el proyecto no desemboca en una aplicación "final", pero proporciona un mecanismo no existente hasta la fecha que posibilita la construcción de un gran número de aplicaciones y despliegues que, a día de hoy, sería complicado desarrollar. En este capítulo se van a describir posibles aplicaciones prácticas del trabajo propuesto que podrían aplicarse al mundo real.

5.1. Marketing

5.1.1. Gestión centralizada de beacons publicitarios en centros comerciales

Una de las posibles aplicaciones del proyecto en *marketing* sería la gestión centralizada de una flota de *beacons* publicitarios desplegada en centros comerciales. Actualmente existen multitud de comercios que utilizan los *beacons* para publicitar sus promociones, ofertas o últimos lanzamientos. El problema que presentan los despliegues actuales radica en la imposibilidad de modificar los contenidos publicitados de forma sencilla, centralizada y remota, ya que en comercios pequeños donde no hay una gran cantidad de *beacons* puede resultar viable, pero en grandes superficies comerciales donde hay cientos de tiendas resulta demasiado tedioso disponer de una persona recorriendo el centro comercial para poder modificar los contenidos que emiten los *beacons* continuamente.

Usando la tecnología que se ha desarrollado en el proyecto, resulta posible modificar de forma centralizada el contenido que ofrecen los *beacons* de todo el centro comercial sin necesidad de desplazarse físicamente cerca del *beacon*. De esta forma, sería posible disponer de una flota de *beacons* por tiendas y pasillos del centro comercial, pudiendo controlar los que se encuentran en

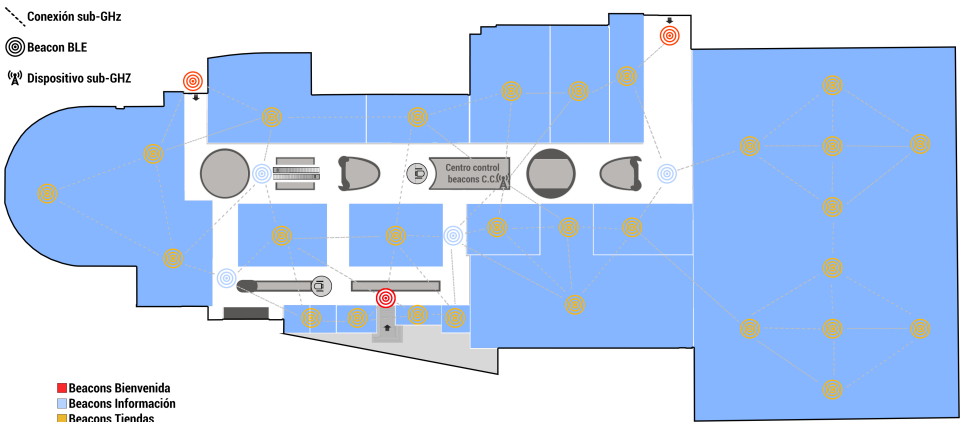


Figura 5.1: Despliegue de *beacons* en un centro comercial.

los pasillos de forma central desde un sólo punto, y pudiendo asignar a cada tienda determinados *beacons* para que si desean gestionarlos ellos mismos puedan hacerlo desde el mostrador de la tienda.

Se puede ver un ejemplo de un despliegue de este tipo en un centro comercial en la figura 5.1.

5.1.2. Beacons móviles con gestión centralizada.

Otra posible aplicación en la misma línea que la anterior sería utilizar la tecnología desarrollada para gestionar una flota de *beacons* móviles de forma centralizada.

A día de hoy, resulta muy común aparcar un vehículo en el centro de una gran ciudad y encontrar a la vuelta el parabrisas cubierto de folletos anunciando comercios de la zona. Ésto genera un gran desperdicio de papel, y un impacto sobre el medio ambiente, dado que la gran mayoría de folletos se desechan automáticamente. La idea sería sustituir estos folletos por *beacons* colocados en vehículos de usuarios que alquilaran sus coches para colocar el dispositivo. Los coches irían circulando por la ciudad y emitiendo *frames* con los contenidos que hayan sido definidos previamente.

Para modificar los contenidos que emiten los *beacons* habría distribuido por la ciudad ciertos puntos con dispositivos con conectividad Sub-GHz y *Ethernet*. Éstos recibirían los cambios por medio de *Ethernet*. Los dispositivos que se encuentran en los vehículos, al detectar que hay uno de estos dispositivos con conectividad Sub-GHz, se unirían a la red que tienen formada y recibirían la publicidad nueva que tienen que emitir mediante *BLE*. De esta forma se puede disponer de una red de *beacons* móviles distribuida por toda la ciudad y que cada cierto tiempo se cambiara que emiten sin necesidad de

Figura 5.2: Despliegue de *beacons* móviles en una ciudad.

tener que hacerlo manualmente.

Se puede ver un ejemplo de un despliegue de *beacons* móviles de este tipo en un ciudad en la figura 5.1.

5.2. Gestión de salas de reuniones

En la mayoría de empresas las salas destinadas para realizar reuniones se gestionan mediante algún programa informático en el que se puede reservar una sala para una determinada hora y añadir a los participantes de la reunión. Para controlar que se aprovechan las salas disponibles y que no hay salas que han sido reservadas, pero debido a una cancelación de la reunión no están siendo ocupadas, aunque en el sistema figuren como tal, existen diversos sistemas mediante los cuales es necesario realizar un *check-in* a la hora de acceder a la sala, de modo que si no se realiza hasta 15 minutos después de la hora de comienzo de la reunión, ésta queda anulada y la sala liberada.

Evoko es uno de estos sistemas que se utilizan con este fin. Es una pantalla táctil que se coloca en la puerta de las salas de reuniones y que muestra la ocupación de la sala. Esta pantalla se encuentra sincronizado con el calendario que tiene asociado la sala en el que se encuentran todas las reuniones

concertadas y los participante que van a asistir. Para realizar el *check-in* tiene incorporado un lector *NFC* que es capaz de leer las tarjetas de los empleados. Además, también permite reservar la sala mediante su pantalla táctil.

Este sistema presenta dos problemas a la hora de su uso. El primero de ellos es el sistema de *check-in*, ya que pueden suceder dos cosas que imposibiliten hacerlo: el primero de ellos, y más obvio, es que el empleado se haya olvidado su tarjeta, por lo que le es imposible realizarlo; el segundo es que para que el sistema reconozca la tarjeta del empleado, ésta debe haber sido de alta con anterioridad. La solución alternativa de *check-in* que propone es introducir el número de empleado en su pantalla táctil, lo que tampoco soluciona completamente el problema debido a que no suele ser un dato que el empleado se sepa y que suele figurar en la tarjeta, por lo que si estamos en el primer caso y el empleado se ha olvidado la tarjeta, es muy probable que tampoco pueda realizar el *check-in* con este método.

El otro problema que tiene el sistema es que no avisa de cuándo acaba la reserva, por lo que si se alarga la reunión más de lo previsto, el siguiente usuario que tenga una reserva en la sala es el encargado de entrar y decir que se ha terminado el tiempo. Por otro lado, si se desea consultar si la hora siguiente está libre para seguir con la reunión, se tiene que salir fuera de la sala que es donde se encuentra la pantalla y mirarlo. Por último, se ha de destacar que estos sistemas suelen ser bastante costosos: el sistema de *Evoko* descrito anteriormente tiene un coste aproximado de 1.500€ por pantalla.

El sistema que se propone para reemplazar sistemas como *Evoko* basados en pantallas es colocar *beacons* gestionados mediante sub-GHz que realicen las funciones de éstas. Estos *beacons* emitirían un *frame* mediante *Eddystone URL* que consistiría en una página web donde se pudiera ver el calendario de la sala. Para realizar el *check-in* se realizaría desde esta misma web introduciendo las credenciales del correo electrónico que se ha utilizado para reservar la sala. Además, se haría uso de notificaciones *push* en el teléfono móvil que ha realizado el *check-in* para avisar del fin de la reserva. De esta manera, si el fin se encuentra próximo, el móvil emitiría una notificación *push* con vibración para llamar la atención de los participantes en la reunión y avisarles de que deben abandonar la sala próximamente. Por último, igual que en las pantallas, se podría realizar reservar *in situ* mediante la misma interfaz web. Además, si se añaden sensores a los dispositivos encargados de realizar la función de *beacons*, es posible utilizarlos para monitorizar distintos parámetros de las salas y así detectar posibles problemas y poder subsanarlos.

Con esta solución, además de añadir funcionalidad a los sistemas que hay en el mercado, subsanamos los problemas que aparecen cuando se usan y con un coste bastante inferior, ya que los dispositivos que se usan para el sistema son mucho menos costosos.

Capítulo 6

Conclusiones, aplicaciones y trabajo futuro

6.1. Conclusiones

A modo de resumen, se va a proceder a enumerar los objetivos que se perseguían al comienzo del proyecto.

El objetivo principal era poder solucionar el mayor problema que tienen los *beacons*, que es tener que estar físicamente próximo a ellos para provisionarlos. A la hora de encarar este problema surgen una serie de retos adicionales que también se consideraron como objetivos: crear una red resistente a caídas y conseguir conectividad punto a punto tanto dentro como fuera de la red desplegada. Por último, se quería añadir funcionalidad a *Contiki* ya que no implementa ningún protocolo para la emisión de datos para *beacons*.

Tras desarrollar el proyecto, podemos decir que el objetivo principal que se perseguía, crear una red de *beacons* controlados de forma remota y centralizada, ha sido cumplido.

Aprovechando la doble banda de las placas que se han utilizado se ha podido desarrollar una red *mesh* resistente a caídas, y gracias al protocolo *6LoWPAN* se ha conseguido tener conectividad punto a punto en toda esta red. Además, con la implementación del *router IP64* junto con el protocolo anteriormente mencionado, *6LoWPAN*, se ha conseguido extender esta conectividad punto a punto fuera de la red, sin importar la topología de red desplegada fuera de nuestra red mallada. Con esto, se puede decir que se han solventado los retos adicionales que suponía el trabajo que se planteó al principio del proyecto: la creación de la red resistente a caídas y la conectividad punto a punto.

Con esta topología desplegada, utilizando *LWM2M* se ha conseguido gestionar la otra interfaz de radio de las placas de 2,4 GHz para emitir en forma de *beacon BLE*.

Ya con ésto, el objetivo principal del proyecto estaría cumplido, se ha conseguido gestionar *beacons* de forma remota mediante sub-GHz sin la necesidad de estar físicamente cerca de ellos.

Por último, para añadir más funcionalidad al sistema se propuso el implementar *Edddystone* en *Contiki*, lo que también se ha podido realizar mediante la implementación del formato *Eddystone URL*.

6.2. Trabajo futuro

Aunque se considera que los objetivos principales del proyecto han sido cubiertos, es cierto que podrían realizarse algunas modificaciones o ampliaciones al proyecto.

A continuación se desarrollan algunas de las ideas que quedarían como trabajo futuro para posibles ampliaciones del proyecto.

La primera ampliación que se podría realizar sería la inclusión de más protocolos para la emisión de datos para *beacons*, ya que únicamente se ha implementado *Eddystone URL* porque se ha considerado que era el más interesante para el proyecto realizado. Así, se podrían implementar el resto de *frames* que nos ofrece *Eddystone* como son *UID*, *TLM* y *EID*.

Además, también sería interesante implementar otros protocolos fuera de *Eddystone*, como por ejemplo *iBeacon* o *AltBeacon*, para dar más flexibilidad a los usuarios a la hora de transmitir datos mediante *beacons BLE*.

También resultaría de interés añadir algún componente de seguridad en LWM2M ya que no se ha introducido ninguna capa de seguridad para, por ejemplo, controlar quién puede acceder a la información de los *beacons*, quién puede controlarlos o qué dispositivos pueden conectarse al servidor *LWM2M*.

Por último, se abordará el desarrollo de un servidor *Leshan* propio, ya que aunque éste es personalizable con los *IPSO Smart Objects* que se van a utilizar, y esta fue la razón por la que no se vio la necesidad de implementar un servidor desde cero, si sería interesante poder personalizarlo cambiando parte la interfaz para hacerla más usable y para que se adaptara específicamente a las necesidades del proyecto.

Capítulo 7

Conclusions and future work

7.1. Conclusions

The main goal of the propoesed work was to solve one of the main drawbacks related with beacons: the necessity of provisioning or configuring them physically from a close location.

When this problem is to be solved, some additional challenges emerge and they were also considered as secondary objectives: to create a lossy resistant network and to be able to have point-to-point connectivity inside and outside the deployed network.

Finally, we added aditional features to Contiki to deal with the broadcasting protocol, not existing yet as a part of the operating system.

Tacking advantage of dual band boards, a lossy resistant network has been developed, and thanks to the *6LoWPAN* protocol, a point-to-point connectivity through the network has been achieved. In addition, given the IP64 router implementation with the aforementioned protocol, the point to point connectivity has been extended outside the mesh network.

With this, we can say the additional challenges raised at the beginning have been achieved: to create a lossy resistant network and to have point-to-point connectivity.

With this topology already deployed and with *LWM2M*, controlling the 2.4 GHz radio interface of the boards to broadcast as a *BLE* beacon has been possible. With this, the overall goal has been achieved: to remotely manage beacons in a centralized fashion through through sub-GHz technology without the need of benig physically near them.

Finally, to add more functionality to Contiki, an implementation of the Eddystone protocol in Contiki was proposed. This was achieved by using the implementation of Eddystone URL version.

7.2. Future work

Although it is considered that all the main objectives proposed at the beginning of the project have been achieved, some modifications or extensions are still posible and are proposed as future work.

The first extension is to include more beacons broadcasting protocols into the project, since only Eddystone URL was implemented (as it was the most interesting for the project). Other beacon broadcasting protocols can be implemented such as iBeacon, AltBeacon or the other data frames that offer Eddystone: Eddystone UID, Eddystone TLM and Eddystone EID.

In addition, it would be interesting to add some security layer into *LWM2M* since no security component has been introduced to control who can access to the beacons information, who can control them or which devices can connect to the server, for example.

Finally, although Leshan permits a personalization of the server with the desired IPSO Smart Objects, it would be interesting to develop a custom Leshan server changing the interface to make it more usable and to be specifically adapted to the needs of the project.

Bibliografía

Alliance, I. *RPL: The IP routing protocol designed for low power and lossy networks*. 2011. Disponible en `http://www.ipso-alliance.org/wp-content/media/rpl.pdf` (último acceso, Junio, 2018).

Alliance, I. *IPSO Smart Objects*. 2018. Disponible en `http://www.ipso-alliance.org/wp-content/uploads/2016/01/ipso-paper.pdf` (último acceso, Junio, 2018).

Alliance, O. M. *Lightweight M2M: Enabling Device Management and Applications for the Internet of Things*. 2014. Disponible en `http://cdn2.hubspot.net/hub/183757/file-610591431-pdf/docs/OMA_Whitepaper_Lightweight_M2M_3-14%5b5%5d.pdf` (último acceso, Junio, 2018).

Foundation, E. *Eclipse Leshan*. 2015. Disponible en `http://www.eclipse.org/leshan/` (último acceso, Junio, 2018).

Google. *Eddystone URL*. 2016. Disponible en `https://github.com/google/eddystone/tree/master/eddystone-url` (último acceso, Junio, 2018).

Google. *Eddystone format*. 2018a. Disponible en `https://developers.google.com/beacons/eddystone` (último acceso, Junio, 2018).

Google. *Google Beacons Platform*. 2018b. Disponible en `https://developers.google.com/beacons/` (último acceso, Junio, 2018).

Instruments, T. *CC1310 SimpleLink Ultra-Low-Power Sub-1 GHz Wireless MCU*. 2016a. Disponible en `http://www.ti.com/lit/ds/symlink/cc1310.pdf` (último acceso, Junio, 2018).

Instruments, T. *CC1350 SimpleLink Ultra-Low-Power Dual-Band Wireless MCU*. 2016b. Disponible en `http://www.ti.com/lit/ds/symlink/cc1350.pdf` (último acceso, Junio, 2018).

INSTRUMENTS, T. *SimpleLink SensorTag Debugger DevPack*. 2018a. Disponible en `http://www.ti.com/tool/CC-DEVPACK-DEBUG` (último acceso, Junio, 2018).

INSTRUMENTS, T. *TI-RTOS: Real-Time Operating System (RTOS) for Microcontrollers (MCU)*. 2018b. Disponible en `http://www.ti.com/tool/TI-RTOS-MCU` (último acceso, Junio, 2018).

SIG, B. *GATT Overview*. 2018. Disponible en `https://www.bluetooth.com/specifications/gatt/generic-attributes-overview` (último acceso, Junio, 2018).

THINGSQUARE. *Contiki OS*. 2006. Disponible en `http://www.contiki-os.org/` (último acceso, Junio, 2018).

TOWNSEND, K. *GAP*. 2015a. Disponible en `https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap` (último acceso, Junio, 2018).

TOWNSEND, K. *Introduction to Bluetooth Low Energy*. 2015b. Disponible en `https://learn.adafruit.com/introduction-to-bluetooth-low-energy` (último acceso, Junio, 2018).

WIKIPEDIA (6LoWPAN). Entrada: "6lowpan".

WIKIPEDIA (Mesh networking). Entrada: "Mesh networking".

WIKIPEDIA (Modulación por desplazamiento de frecuencia gausiana). Entrada: "Modulación por desplazamiento de frecuencia gausiana".

WIKIPEDIA (System on a chip). Entrada: "System on a chip".

Acrónimos

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
ACK	Acknoewledgement
API	Application Programming Interface
ATT	Attribute Protocol
BLE	Bluetooth Low Energy
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
DHCP	Dynamic Host Configuration Protocol
DIO	DODAG Information Object
DIS	DODAG Information Solicitation
DODAG	Destination-Oriented Directed Acyclic Graph
DTLM	Datagram Transport Layer Security
DTLS	Datagram Transport Layer Security
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GPU	Graphics Processing Unit
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Procotol version 4
IPv6	Internet Procotol version 6

IPSO	Internet Protocol for the networking of Smart Objects
ISM	Industrial, Scientific and Medical
JSON	JavaScript Object Notation
LAN	Local Area Network
LTV	Type-Length-Value
LNN	Low-Power and Lossy Network
LoWPAN	Low power Wireless Personal Area Networks
LWM2M	Lightweight Machine To Machine
M2M	Machine To Machine
MAC	Media Access Control
MCU	Microcontroller Unit
MQTT	Message Queue Temetry Transport
NFC	Near Field Communication
OMA	Open Mobile Alliance
O/I/R	Object Instance Resource
OTA	Over-The-Air
PDU	Protocol Data Unit
POSIX	Portable Operating System Interface
RAM	Random Access Memory
ROM	Read-Only Memory
RPL	Routing over Low Power and Lossy Networks
RSSI	Received Signal Strength Indicator
RTOS	Real Time Operating System
SoC	System On Chip
SPI	Serial Peripheral Interface
TI	Texas Instruments
TI RTOS	Texas Instruments Real Time Operating System

UDP User Datagram Protocol

URL Uniform Resource Locator

UUID Universally Unique Identifier

WLAN Wireless Local Area Network

WMAN Wireless Metropolitan Area Network

WPAN Wireless Personal Area Network

WWAN Wireless Wide Area Network

XML Extensible Markup Language